UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

PORTAL DE MONITORIZACIÓN Y DIMENSIONADO DE INFRAESTRUCTURAS DE *CONTACT CENTER MULTITENANT*

Aitor Hernández Galindo

Tutor: Sérgio Filipe Alves Ribeiro de Almeida

Ponente: Jorge Enrique López de Vergara Méndez

JUNIO 2017

PORTAL DE MONITORIZACIÓN Y DIMENSIONADO DE INFRAESTRUCTURAS DE *CONTACT CENTER MULTITENANT*

AUTOR: Aitor Hernández Galindo
TUTOR: Sérgio Filipe Alves Ribeiro de Almeida

Dpto. de Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2017

Resumen

En una plataforma de *contact center multitenant* (múltiples *contact centers* virtuales ofrecidos en modelo *as a Service* a partir de una infraestructura única/distribuida) es crucial poder monitorizar la cantidad de recursos que cada cliente está consumiendo y, por tanto, registrar y analizar estos datos nos puede permitir tener un mayor control en la manera de gestionar la infraestructura. Pudiendo saber de esta manera, por ejemplo, cuántos clientes pueden ser alojados en la infraestructura actual, o incluso poder variar los recursos asociados a la infraestructura dependiendo del consumo de recursos que los clientes están realizando.

Actualmente, dentro de la plataforma de *contact center* de Collab, no existe manera de poder obtener información sobre el consumo que sus componentes estas realizando sobre la infraestructura, y cuanto menos, existe la posibilidad de relacionar esta información con los valores de negocio manejados por la plataforma de *contact center*.

Por tanto, este Trabajo Fin de Grado viene a ofrecer una solución a este problema, con la implementación de una aplicación de consola que va recogiendo periódicamente los *Key Performance Indicators* (KPIs) de la plataforma de *contact center* e información del consumo de recursos de los componentes, para realizar un mapeo entre ellos y poder saber de qué manera está afectando cada instancia al consumo de recursos en la infraestructura.

Además, para poder consultar la información generada por la aplicación de consola nombrada anteriormente y para poder obtener información en tiempo real del consumo de recursos que se está produciendo en cada componente, en el ámbito de este TFG ha sido desarrollada una aplicación web, donde incluso es posible realizar simulaciones para saber si un nuevo cliente puede ser alojado en la infraestructura actual en base a sus estimativas de utilización de la plataforma.

Para obtener información del consumo de recursos en los componentes del *contact center*, ha sido realizada una integración con la solución de monitorización de red llamada PRTG, la cual dispone de una API REST bastante completa e intuitiva, que facilita el proceso de integración.

El hecho de disponer de los datos mencionados anteriormente es muy relevante sobre todo en implantaciones soportadas por proveedores de servicios en la nube, los cuales realizan facturación en base al uso que se ha dado los recursos contratados, ya que estos datos van a permitir optimizar la infraestructura que soporta la plataforma de *contact center*.

Para finalizar, he desplegado un ambiente de pruebas con el objetivo de generar carga de manera progresiva sobre los componentes del *contact center* para validar cuál era su comportamiento con los diferentes niveles de carga. Para la obtención de los resultados he utilizado la aplicación web desarrollada en el ámbito de este TFG, y tanto estos resultados como las conclusiones tomadas, pueden ser consultados en el capítulo “Integración, Pruebas y Resultados” de la presente memoria.

Abstract

On a multitenant contact center platform (multiple virtual contact centers offered as a service from a single/distributed infrastructure) it is utterly important being capable of monitoring and managing each of tenant's resources consumption. These managed services are often performed by means of logging and analyzing the different activities, as well as associated key performance indicators, of each of the tenants platform components, which will then lead to a better quality of service.

An improved QoS (Quality of Service) comes with the understanding of the consumption patterns. Providers can then make decisions, and plans, on how to better allocate infrastructure resources, which will have direct impact on the sizing and functionalities supported by each of the tenants. Business wise, being able to fully understand how resources are being consumed also enables organizations to reduce waste, costs as well as easing IT Governance.

Following this train of thought this Bachelor Thesis not only focuses on the inability of Collab's multitenant contact center platform in extensively reporting/logging the previously mentioned data, and KPIs, but also on a solution to overcome it. Collab is a software house that specializes on contact center solutions and is now doing the market shift to move its business to cloud born solutions and hence the relevance of the chosen theme. The cloud paradigm means that not only these enhancements are necessary for the day-to-day management of the service in the organization but they are also deemed required for proper customer invoicing because part of the Software as a Service business includes the reflection of the hosting/platform costs to the customers.

The solution I have designed to tackle the aforementioned problematic consists on the deployment of a console application which is capable of gathering usage statistics and KPIs overtime. These will then allow a proper summary and vision of the consumed resources, per tenant, that have direct impact on the contact center infrastructure overall performance.

I have also decided to complement this solution with the development of a web app where one can simulate the impact that a new tenant would cause on the existing infrastructure, given the tenant's requirements; or if it is required to allocate more resources to the infrastructure so that a new tenant can be provisioned and hosted.

In furtherance of Collab's contact center consumption data and KPIs, an integration with the PRTG network-monitoring tool has been performed. This tool facilitates the integration process with a very intuitive and complete REST API, which enables me to fetch some of the required data and KPIs.

To conclude, a trial and load test environment has been set up so that I could, progressively, increase the load on the contact center's platform components in order to validate their behavior under different loading conditions. The carried out study results, and conclusions, can be consulted on the chapter *"Integración, Pruebas y Resultados"* of this Bachelor Thesis.

Palabras Clave

Contact center, *Key Performance Indicator*, Componente, Agente, PRTG, Monitorización, SIPp, *Agent Wrapper*, Conector, Sensor, Dispositivos, Instancia.

Keywords

Contact center, Key Performance Indicator, Component, Agent, PRTG, Monitoring, SIPp, Agent Wrapper, Connector, Sensor, Device, Tenant.

Agradecimientos

A mis padres por el apoyo incondicional.

A Joana, por el apoyo y la comprensión, ya que el tiempo invertido en el TFG sale de momentos en los cuales no la he podido acompañar.

A Sérgio por la oportunidad profesional que me ha brindado.

Tabla de Contenidos

Índice de Figuras

Índice de Tablas

Glosario de Acrónimos

AAA	Authentication, Authorization and Accounting
API	Application Programming Interface
DDI	Direct Dial-in
DNIS	Dialed Number Identification Service
DTMF	Dual-Tone Multi-Frequency Signaling
DoS	Denial-of-Service
IDE	Integrated Development Environment
IM	Instant Messaging
IP	Internet Protocol
IVR	Interactive Voice Response
KPI	Key Performance Indicator
MVC	Model–View–Controller
NAT	Network Address Translation
QA	Quality Assurance
QoS	Quality of Service
RAM	Random-Access Memory
REST	Representational State Transfer
SBC	Session Border Controller
SDK	Software Development Kit
SIP	Session Initiation Protocol
TCP	Transmission Control Protocol
TFG	Trabajo Fin de Grado
TLS	Transport Layer Security
UDP	User Datagram Protocol
WebRTC	Web Real-Time Communication

1 Introducción

1.1. Motivación

El tema escogido para este TFG vino por la búsqueda de un asunto que estuviese relacionado, en la medida de lo posible, con el trabajo que yo desarrollo en Collab. Ya que mi función en esta empresa es *Sales Engineer*, y no está dentro de mis tareas la realización de desarrollos o investigaciones, tuve que buscar, junto a mi jefe, que también es el tutor de este TFG, un tema que nos interesase y que pudieses aprovechar en el día a día.

El tema escogido está relacionado con el dimensionamiento, en términos de arquitectura de la plataforma de *contact center* que Collab comercializa y el estudio de los consumos que la misma plataforma genera sobre la arquitectura donde está instalada.

Con información objetiva y siempre actualizada de los consumos de la plataforma, independientemente de si la plataforma sufre actualizaciones, vamos a conseguir realizar dimensionamientos más adecuados para nuestros clientes, lo cual sí que es una de las responsabilidades asociadas a mi cargo.

Al fin y al cabo, esto es muy importante para nuestro departamento, ya que nos permite tener independencia del equipo del equipo de QA, el cual era responsable por esta tarea, no siendo una prioridad dentro de sus funciones, lo que hacía con que, en los últimos tiempos tuviésemos que trabajar con datos desactualizados.

Además, considerando la oferta que está lanzando actualmente Collab, de *contact center* en la nube, los desarrollos realizados en el ámbito de este TFG pueden ser aprovechados para futuros desarrollos, los cuales son detallados en el capítulo “Trabajo Futuro”.

1.2. Objetivos

El objetivo de este TFG es realizar una aplicación orientada a la obtención de consumo de recursos de la solución de *contact center* de Collab, en las infraestructuras donde es instalada. Esta solución podrá ser incorporada, en el futuro, en la aplicación de administración del *contact center* para la obtención de alarmística y podrá servir como calculadora de número de entidades (agentes, puertos de IVR, puertos de grabación, …) que es posible crear dentro la plataforma de *contact center* con la infraestructura actual.

Por otro lado, también es objetivo de este TFG la mudanza en la manera como los datos de consumo de recursos, por parte de los componentes del *contact center*, son mostrados.

Permitiendo una visualización consolidada de todos los sensores que hacen referencia a cada componente del *contact center*.

1.3. Organización de la Memoria

La memoria consta de los siguientes capítulos:

- **Introducción**: exposición de las motivaciones que me llevaron a realizar este TFG y cuáles son los objetivos a alcanzar;
- **Estado del Arte**: descripción de los puntos en que la solución diseñada y desarrollada en el ámbito de este TFG, puede aportar un valor adicional a las implementaciones actuales realizadas por Collab y el favor en términos de conocimientos que esta solución puede dar al departamento de pre-venta;
- **Diseño**: explicación detallada de cuál fue el diseño escogido, tanto para la Aplicación de Consola, como para la Aplicación Web, y explicación del diseño de los conectores para la plataforma de *contact center* y para PRTG, utilizados por las dos aplicaciones nombradas anteriormente;
- **Desarrollo**: métodos de implementación utilizados para desarrollar las aplicaciones y conectores descritas en el capítulo de Diseño;
- **Integración, Pruebas y Resultados**: descripción del ambiente de pruebas creado y las diferentes máquinas que lo componen. Explicación de las pruebas realizadas sobre las plataformas, exposición de los valores obtenidos en las pruebas e interpretación de los resultados;
- **Conclusiones y Trabajo Futuro**: reflexión sobre las lecciones aprendidas, conclusiones de las pruebas realizadas y desarrollos que pueden ser realizados partiendo de la solución presentada en este TFG;
- **A**
  Componentes de la Plataforma de Contact Center: lista de componentes de la plataforma de *contact center* y su descripción;
- **B**
  Lista de Sensores Monitorizados a partir de PRTG**:** lista completa de todos los sensores que pueden ser manejados por el conector desarrollado para PRTG.

2
Estado del Arte

2.1. Necesidades Detectadas en la Plataforma de Contact Center

La plataforma de *contact center* de Collab es una solución basada completamente en el protocolo SIP, la cual permite el atendimiento al cliente por diferentes puntos de contacto, como son voz, vídeo, chat, e-mail y redes sociales (Facebook y Twitter). Esta plataforma además de ser multicanal, es multi-instancia (*multitenant*), por tanto, en una misma instalación de la plataforma, es posible albergar varios clientes, los cuales, aunque comparten recursos y todos interactúan con los mismos componentes de la plataforma, existe una separación a nivel de base de datos para asegurar la independencia en termino de datos de cada cliente [1].

Esta plataforma de *contact center*, puede ser instalada en una única máquina cuando la solución va a soportar un número pequeño de agentes, y puede escalar de manera horizontal, con balanceo de carga, de manera que va aumentando el número de agentes que son soportados por la plataforma. De hecho, el sistema puede alcanzar un nivel de distribución muy elevando, lo que dificulta mucho su monitorización.

Actualmente, en la plataforma de *contact center* no existe ninguna aplicación en la que pueda ser monitorizada la infraestructura sobre la que están instalados los componentes del contact center, y cuanto menos la relación de esta información con valores de negocio. Por tanto, uno de los objetivos de este TFG es cubrir esta necesidad con la creación de una aplicación que muestre el consumo que realizan los componentes de la plataforma de contact center y por cada uno de esos componentes, conseguir saber cuál, fue el consumo realizado por cada instancia.

Además, también se marca como objetivo, la creación de una interfaz de monitorización de la infraestructura directamente relacionada con las características específicas de la plataforma de *contact center*. Al contrario que las soluciones de monitorización genéricas, donde se tratan por separado los valores de procesos de Windows, accesos a puertos, conexión SIP, etc., lo que se pretende con esta solución es dar una visión global de todos los sensores que están relacionados con cada uno de los componentes del *contact center*.

Por otro lado, actualmente en el departamento de pre-venta no tenemos siempre valores actualizados de los consumos generados por cada uno de los componentes del *contact center*, por tanto, también es objetivo de este TFG el poder obtener conocimiento sobre las herramientas de prueba que nos permiten obtener estos valores orientativos y poder realizar pruebas para analizar como escala la plataforma y poder mejorar la manera como realizamos los dimensionamientos a la hora de proponer nuestro producto a los clientes.

2.2. Decisiones Tecnológicas

Para ayudar a convertir en realizad una solución que cubra las necesidades expuestas en el punto anterior, nos hemos apoyado en la solución PRTG, la cual está destina a realizar monitorizaciones de red.

Los principales motivos de esta decisión son la madurez de la solución, con un gran número de funcionalidades disponibles, la simplicidad a la hora de manejarla y la cantidad documentación disponible, siendo este último punto de vital importancia a la hora de intentar aprender a interactuar e integrar con una solución desconocida. Pero, sobre todo, el factor más importante que llevó a la selección de PRTG es su API REST, la cual permite una integración simple e intuitiva. Además, PRTG, aun siendo una solución empresarial, tiene una versión free que permite la creación de hasta 100 sensores, lo cual es un punto positivo a la hora de crear un ambiente te desarrollos o de pruebas, ya que no tiene costes asociados.

3
Diseño

3.1. Sumario del Diseño

La presente solución está compuesta por dos aplicaciones, la primera es una aplicación de consola, la cual pude ser convertida en servicio Windows a la hora de implementar en un cliente y la segunda es una aplicación web que consume la información generada por la aplicación de consola y utiliza las interfaces desarrolladas en el ámbito de este TFG para mostrar información proveniente de PRTG y de la plataforma de *contact center*.

La aplicación de consola tiene como principal función, el cruzamiento de la información proveniente de la plataforma de *contact center* y de PRTG. Este cruzamiento es realizado por un módulo denominado Calculadora, el cual tiene conocimiento de cuáles son los indicadores (KPIs), cuya variación de valor provoca cambios significativos en el consumo realizado por el *contact center*, y utiliza esta información para calcular los valores unitarios de consumo asociados al indicador.

Para entender mejor como se relacionan los diferentes componentes que hacen parte de esta solución, se incluye el diagrama de alto nivel representado en la “Figura 3.1”, cuyo objetivo es, que de manera introductoria, el lector tenga noción visual de la solución final que ha sido diseñada en el ámbito de este TFG.

En la siguiente figura es representada una implementación de OneContact con varios clientes/instancias, siendo soportados por una misma infraestructura. También está representado un servidor de bases de datos, el cual además de dar soporte a la plataforma de *contact center*, va a servir para gestionar datos en el ámbito de la solución de este TFG. Junto a lo ya comentado en el inicio de este parágrafo, lo cual simplemente define una implementación de OneContact típica, han sido incluidos un servidor de PRTG y otro servidor que incluye los desarrollos realizados en el ámbito de este TFG.

Otra información importante que se puede obtener de la siguiente figura, es la manera como los diferentes componentes representados interactúan, donde podemos ver que el conector con OneContact utiliza un socket TCP y el conector con PRTG que realiza pedidos a la API a través de HTTP. Esta información suministrada por la API de PRTG, como está representado en la figura, ha sido previamente obtenida por PRTG consultando los diferentes componentes monitorizados, a través del protocolo CIM-XML sobre HTTP.

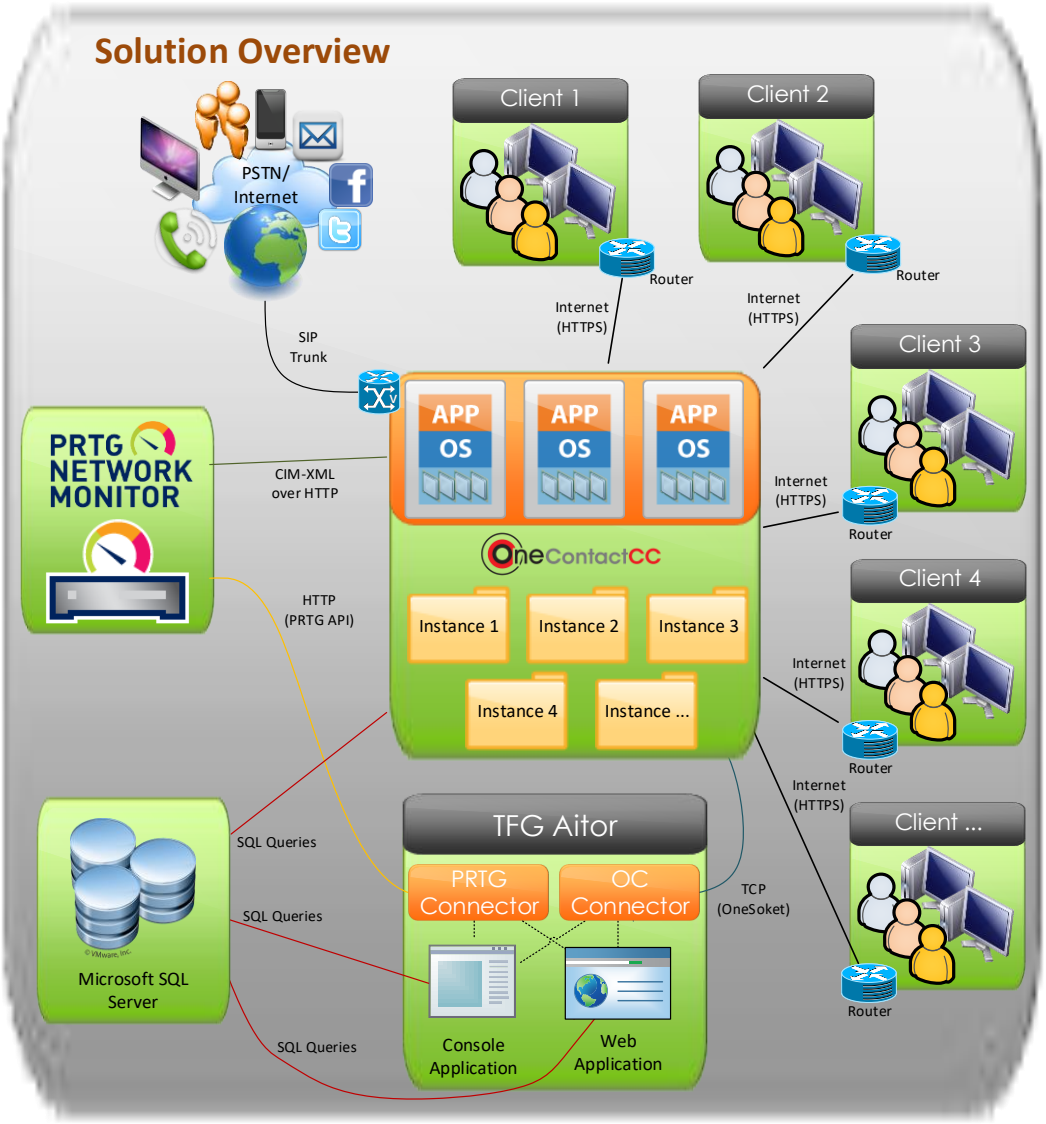

Figura 3.1: Visión General de la Solución

3.2. Integración con PRTG

PRTG posee una API basada en web "RESTful" que permite a aplicaciones externas acceder a la base de datos de monitorización y manipular objetos dentro de la base de datos de PRTG. Por ser una API "RESTful", las consultas son realizadas a través de pedidos HTTP/HTTPS, más concretamente pedidos GET, los cuales retornan documentos XML, JSON, CSV y HTML [2]. Esta API es la que ha sido utilizada en el ámbito de este TFG para realizar la integración con PRTG.

La API de PRTG ofrece las siguientes funcionalidades:

- Autenticación;
- Métodos para la obtención de información de objetos en tiempo real;
- Métodos para la obtención de información histórica de sensores;

- Métodos para la manipulación de objetos.

El módulo de integración desarrollado en el ámbito de este TFG está compuesto por 3 clases, PRTGConnector, PRTGDevice y PRTGSensor, las cuales son introducidas a continuación:

- **PRTGConnector**: contiene todos los métodos de interacción con la API de PRTG;
- **PRTGSensor**: define la entidad sensor dentro del universo PRTG, la cual es una composición jerárquica de varias clases, para realizar la distinción de los diferentes tipos de sensores;
- **PRTGDevice**: define la entidad *device* dentro del universo PRTG, la cual en rasgos generales es una colección de PRTGSensors.

3.2.1. PRTGConnector: Interacción con la API de PRTG

La clase estática PRTGConnector sirve como interfaz para interactuar con la API de PRTG, la cual, como comentado anteriormente es una API "RESTful". Por tanto, el primer método que se hace necesario en esta clase, es un método auxiliar al resto de métodos de la clase, que a partir de un URL previamente construido, realiza pedidos GET de manera genérica y obtiene la respuesta a los pedidos asignado ese valor a una *string*.

Esta clase tiene los siguientes campos:

- ***Username***: nombre de usuario utilizado para autenticación en los pedidos a la API de PRTG;
- ***Passhash***: código hash utilizado para autenticación en los pedidos a la API de PRTG y que es obtenido a partir de un otro pedido a la API, el cuál es explicado en el próximo parágrafo;
- **URL**: el valor para este campo es obtenido del fichero de configuración de la aplicación, y hace referencia al URL de la API de PRTG contra el cual se realizan los pedidos.

El método de inicialización de esta clase también realiza las funciones de autenticación en PRTG, ya que, realiza un pedido a la API pasando un nombre de usuario y una contraseña, el cual retorna un código hash. Este código hash retornado por la API, denominado por PRTG como *passhash*, es utilizado de aquí en adelante junto al nombre de usuario en concepto de autenticación para cualquier nuevo pedido realizado a la API PRTG.

Aun hablando en términos de autenticación, a parte del proceso de inicialización, ya que el *passhash* es necesario para cualquier pedido que queramos realizar a PRTG, en la aplicación web, el hecho de hacer logout, básicamente pone a *null* este campo dentro de la clase del conector.

Entrando en detalle en los puntos de interacción con PRTG, a continuación, son descritos los 3 métodos que permiten la obtención de datos de PRTG:

RecentResult:

Gracias a este método es posible obtener los últimos "n" valores de un sensor. Los sensores de PRTG tienen un identificador asociado, el cual es utilizado para realizar el pedido a la API. Este pedido nos retorna un documento JSON, que entre otros campos incluye el intervalo de fecha y hora correspondiente a esa muestra y los valores medios del sensor durante el periodo de tiempo indicado por el campo anterior.

En el ámbito de este TFG sólo trabajamos con el valor más reciente del sensor, por tanto, nuestro "n" siempre va a ser igual a 1, y para que este "n" sea exactamente el valor más reciente debemos pedir a PRTG la ordenación de la respuesta por fecha de manera decreciente. El motivo de sólo pedir el valor más reciente del sensor, es que este método va a ser utilizado por la aplicación de consola, la cual siempre va a estar consultando por el valor más reciente a PRTG, con el cual realiza ciertos cálculos que serán explicados posteriormente. Por otro lado, este método también es utilizado para obtener valores de monitorización, los cuales serán mostrados en la aplicación web.

Ya que cada tipo de sensor tiene campos diferentes, para generalizar al máximo este método, cuando es invocado debe ser indicada la clase esperamos que nos retorne, la cual está tipificada a una clase genérica llamada PRTGSensorValues, la cual será descrita más adelante. Esto es necesario ya que la API de PRTG retorna un documento JSON el cual des-serializamos utilizando la librería Newtonsoft.Json [3]. Dicho con otras palabras, convertimos el documento JSON en un objeto del tipo indicado en la invocación del método RecentResult, y para que esta conversión no cree excepción, los campos del tipo indicado en la invocación, deben coincidir con las palabras llave del documento JSON.

Un punto importante a referir sobre el proceso de conversión del retorno de la API a objeto del tipo PRTGSensorValues, es que para realizar el mapeo entre los nombres de los campos de cada sensor en PRTG y los nombres de los campos de las clases .NET, por causa de restricciones en los nombres del lado de .NET, es necesario pre-procesamiento del documento JSON retornado por la API. Este proceso para por los siguientes pasos, los cuales están basados en decisiones tomadas en el proceso de diseño:

- Remplazo del carácter '/' por '_' dentro del documento JSON;
- Remplazo del *string* " (" por el carácter '_' dentro del documento JSON;
- Eliminación del carácter ')' dentro del documento JSON;
- Eliminación de los espacios dentro del documento JSON;
- Eliminación del carácter '%' dentro del documento JSON;
- Remplazo del carácter '#' por el *string* "*Number*" dentro del documento JSON.

Con esto conseguimos obtener, con único método, la información de cualquier tipo de sensor de PRTG y rellenar con esa información una clase .NET para poder manejar esta información manera mucho más simple.

GetDevices:

El método *GetDevices* tiene como funcionalidad pedir a través de la API de PRTG la información referente a los dispositivos donde están instalados los diferentes componentes de la plataforma de *contact center*. Para ello es utilizado un método especifico del API PRTG, que al igual que los otros, retorna un documento JSON, el cual es recorrido obteniendo documentos JSON individuales por cada dispositivo, los cuales son des-serializados, o sea convertidos en objetos del tipo *PRTGDevice* y añadidos a una lista, que sirve como retorno de este método.

GetSensors:

Con el método *GetSensors* podemos obtener la lista de sensores que están configurados en un dispositivo, a partir del identificador de dicho dispositivo. Para obtener esta información es utilizado otro método de la API PRTG que retorna un documento JSON, el cual es recorrido obteniendo los documentos JSON individuales por cada sensor, los cuales son des-serializados, o sea convertidos en objetos del tipo *PRTGSensor* y añadidos a una lista, que sirve como retorno de este método.

En los 3 métodos detallados anteriormente, para el proceso de construcción del URL que realizar la invocación de la API de PRTG, la cual en el ámbito de este TFG siempre nos retorna una lista o tabla de entidades, es necesario indicar los siguientes parámetros [4]:

- ***Content*:** indica la información que esperamos que nos retorne el pedido. Dentro del ámbito del TFG, han sido utilizados los siguientes valores para este parámetro:
  - *Values*: para el método RecentResult;
  - *Devices*: para el método GetDevices;
  - *Sensors*: para el método GetSensors.
- ***Output***: define el formato de la información retorna por la API. El valor utilizado en el ámbito de este TFG es JSON, por la facilidad que este formato ofrece en términos de tratamiento de la información;
- ***Columns***: donde definimos los campos que queremos que nos retorne en relación a la entidad indicada en el parámetro *content*;
- ***Count***: sirve para indicar el número de entidades del tipo indicado en el parámetro *content* que queremos que sean incluidas en el documento retornado;
- ***Id***: este identificador hace referencia a la entidad por la cual estamos realizando la consult. Puede ser, por ejemplo, el identificador del dispositivo o del sensor;
- ***Sortby***: en este parámetro indicamos porque columnas, de las referidas en el parámetro *columns* queremos ordenar los resultados de nuestro pedido. Para indicar el sentido de la ordenación, colocamos un signo negativo antes del nombre de la columna en el caso de que deseemos orden decreciente y en caso contrario únicamente es indicado el nombre de la columna;
- ***Username***: como ya referido anteriormente, es necesario realizar autenticación en cada pedido realizado a la API de PRTG, por tanto, es necesario indicar el nombre de usuario.
- ***Passhash***: este valor es obtenido realizando un pedido a la API de PRTG y junto con el *username* tiene función de autenticación.

3.2.2. PRTGSensor: Representación de un Sensor de PRTG

Para la representación de un sensor de PRTG dentro la aplicación diseñada en el ámbito de este TFG ha sido necesario el diseño de una jerarquía de clases, con el objetivo de generalizar lo máximo posible esta entidad. La clase raíz es PRTGSensor, la cual está definida por los siguientes campos:

- ***Objid***: identificador del sensor dentro de la base de datos de PRTG;
- ***Sensor***: nombre del sensor dentro de la base de datos de PRTG;
- ***Type***: tipo del sensor;
- ***Values***: este campo, del tipo PRTGSensorValues, alberga los valores del sensor. La clase abstracta PRTGSensorValues será explicada con más detalle a continuación.

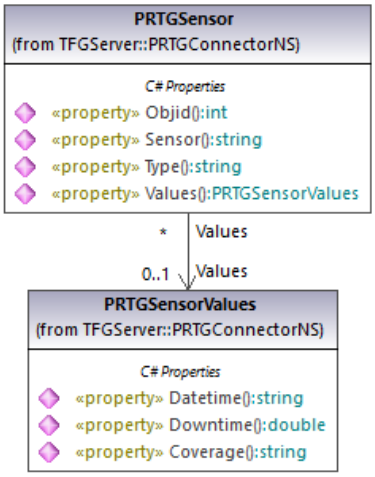

Figura 3.2: Diagrama de clases PRTGSensor -> PRTGSensorValues

En el proceso de inicialización de PRTGSensor es identificado el tipo de sensor para realizar la correcta inicialización del campo *Values*, es añadido a la base de datos de nuestra aplicación como siendo un sensor que está dentro de nuestro universo de monitorización, ya que podemos no querer monitorizar todos los sensores de un dispositivo, y es realizado un primer pedido a la API de PRTG para obtener los valores del campo *Values*.

Para asignar los valores del sensor a un objeto del tipo PRTGSensor, ha sido creada una una clase abstracta denominada PRTGSensorValues, está clase tiene los campos comunes a todos los pedidos de valores realizados a la API de PRTG:

- ***Datetime***: intervalo de fechas/hora a que corresponde ese valor;
- ***Downtime***: porcentaje de tiempo que ese valor estuvo dentro de los valores de error configurados en PRTG;
- ***Coverage***: es la calidad de los datos, en porcentaje, dada por la cantidad de mediciones que dieron lugar a ese valor durante el periodo de tiempo indicado en *datetime*.

A partir de PRTGSensorValues existen un conjunto de clases que heredan de esta y que definen todos los sensores que tenemos en PRTG. En la siguiente figura está representado el diagrama de clases con todas las herencias directas de PRTGSensorValues:

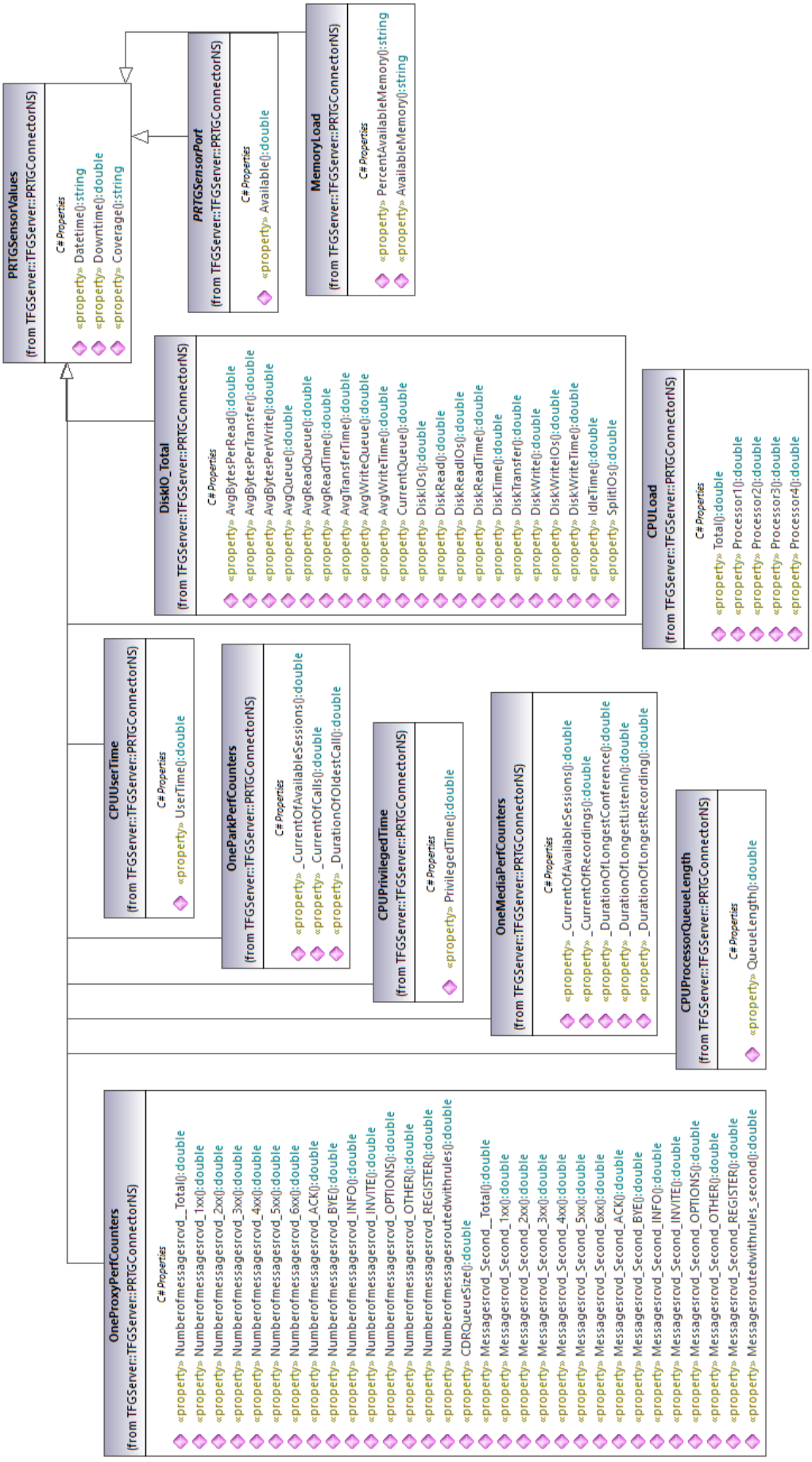

Figura 3.3: Diagrama de Clases con Herencia Directa de PRTGSensorValues [1/2]

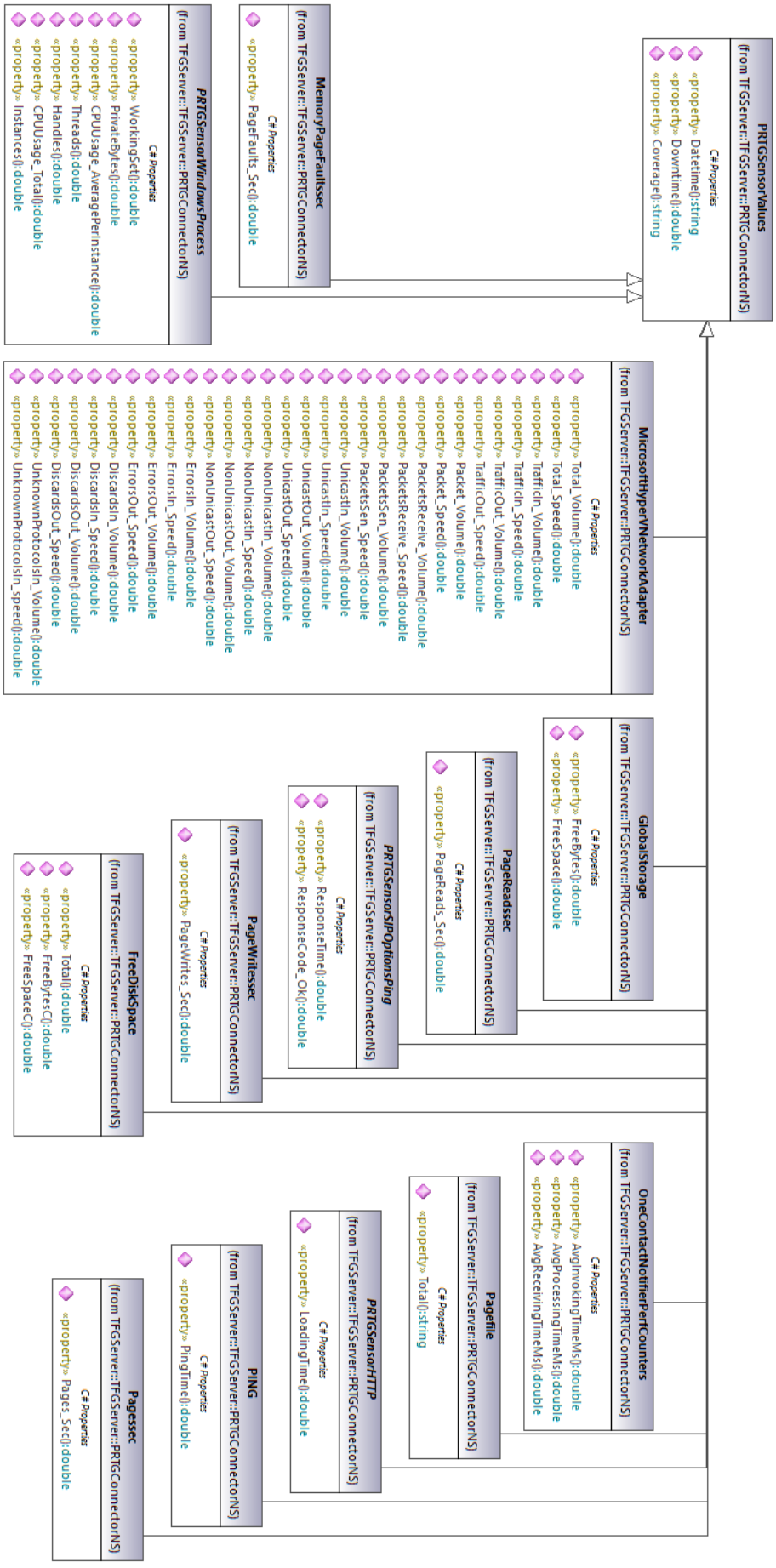

Figura 3.4: Diagrama de Clases con Herencia Directa de PRTGSensorValues [2/2]

Algunas de las clases representadas en el diagrama de clases anterior son abstractas ya que sobre ese tipo de sensor podemos tener varios sensores diferentes, los cuales comparten características. En las siguientes figuras son representados los diagramas de clases individuales, de las clases abstractas que heredan de PRTGSensorValues:

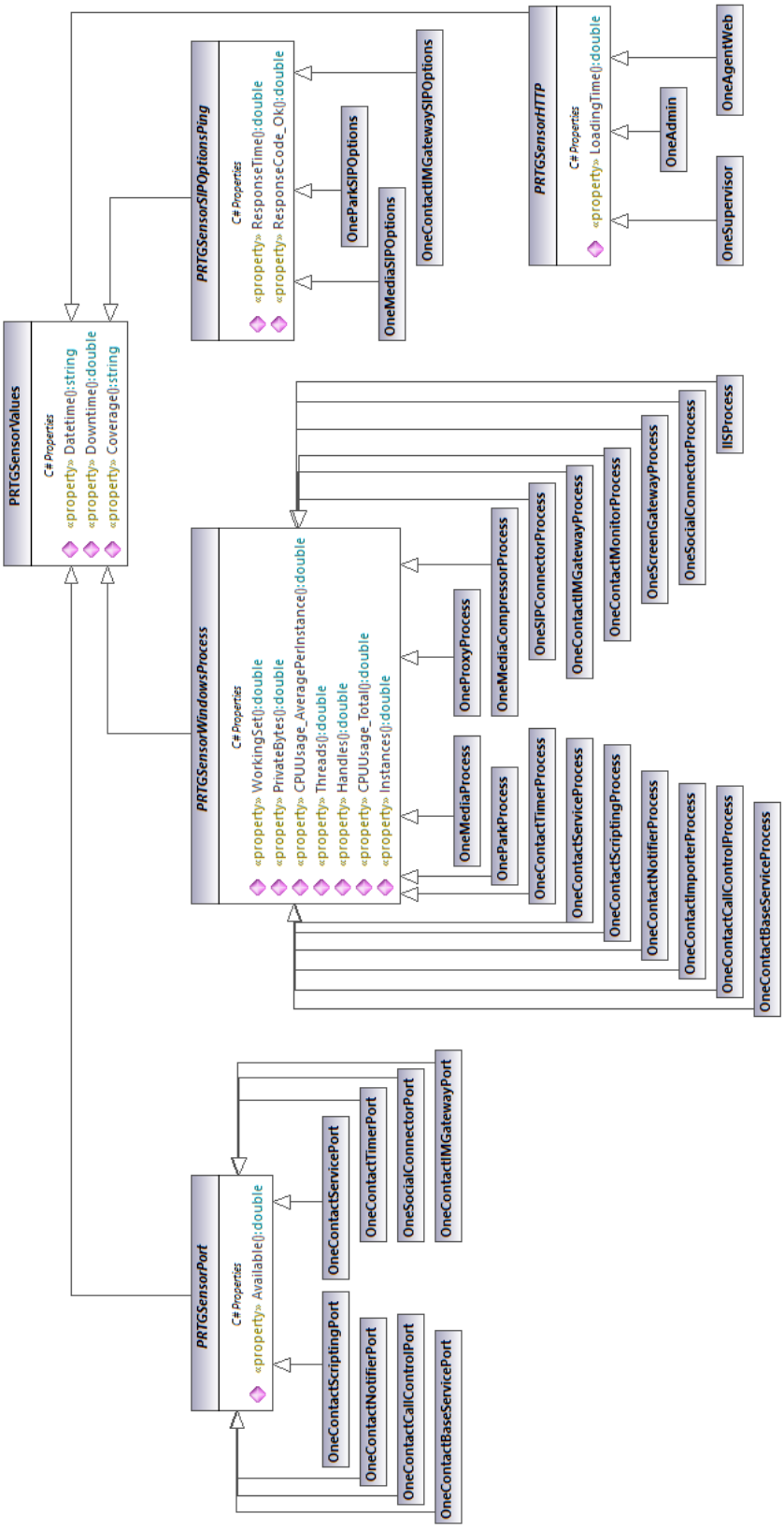

Figura 3.5: Diagrama de Clases Abstractas que Heredan de PRTGSensorValues

3.2.3. PRTGDevice: Representación de un Dispositivo de PRTG

Con el objetivo de poder gestionar los datos de los diferentes dispositivos donde están instalados los componentes de la solución de *contact center*, fue diseñada la clase PRTGDevice. Cada dispositivo tiene un conjunto de sensores configurados, por tanto, a grandes rasgos, esta clase es una agrupación de sensores (PRTGSensor).

La clase PRTGDevice está compuesta por los siguientes campos:

- ***Objid***: identificador del dispositivo dentro de la base de datos de PRTG;
- ***Group***: grupo de PRTG en el que está incluido el dispositivo;
- ***Device***: nombre del dispositivo dentro de la base de datos de PRTG;
- ***Sensors***: lista de sensores que están configurados en el dispositivo.

En el momento que se crea un nuevo objeto de este tipo en la aplicación, es realizado un pedido a la API de PRTG para obtener la lista de sensores configurados en el dispositivo y es realizado otro pedido para obtener los valores de todos los sensores que forman parte de esta lista.

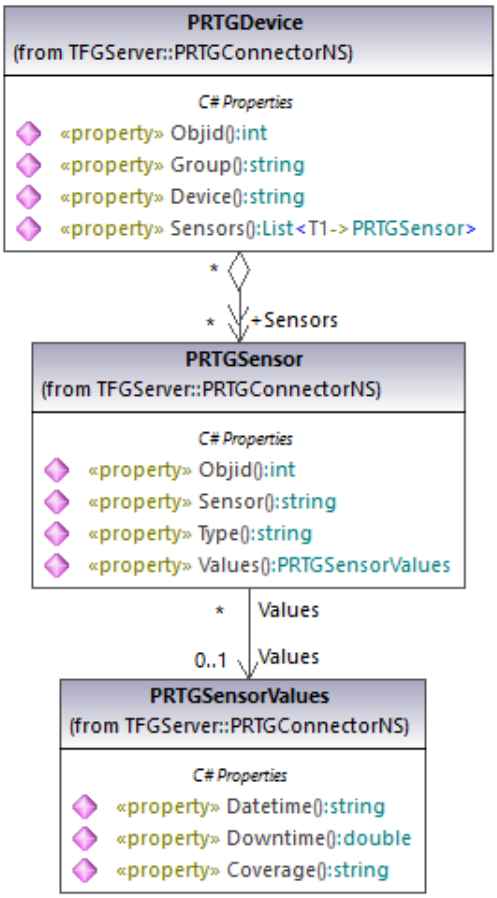

Figura 3.6: Diagrama de Clases de PRTGDevices -> PRTGSensor -> PRTGSensorValues

3.3. Integración con la Plataforma de Contact Center

Para la integración con la plataforma de *contact center* ha sido utilizado el SDK que la misma plataforma ofrece. Más concretamente ha sido utilizado el StatisticsService del SDK, el cual proporciona información en tiempo real de los KPIs del *contact center* [5].

Como base de todas las interacciones realizadas por este SDK está la invocación de un *web service* denominado BaseService, con el cual se puede interactuar para pedir cualquier dato accesible de la plataforma de *contact center*. Este BaseService es configurado en la inicialización del objeto StatisticsService, indicando la instancia del *contact center* de que queremos obtener datos, con el propósito de poder invocar los métodos de StatisticsService y que retorne adecuadamente la información solicitada.

Los métodos implementados en la clase estática OCConnector para obtención de datos de StatisticalDataProvider, son invocados por la clase OCKPI, la cual define la entidad KPI del *contact center*.

3.3.1. Obtención de Datos con StatisticalDataProvider

Los métodos diseñados dentro de la clase OCConnector realizan pedidos a la clase StatisticsService del SDK de la solución del *contact center* a través de llamadas al método RunStatistical. Para realizar estos pedidos es necesario indicar a que StatisticalDataProvider queremos realizar el pedido y cuál es el filtro (SDPRowsIDsDefinition) que queremos realizar sobre los datos de ese StatisticalDataProvider.

A continuación, están listados los métodos que han sido diseñados, en base a la necesidad de obtención de información sobre los indicadores del contact center que requerimos manejar dentro de la aplicación:

- **GetStartedServices**: método para obtener información sobre todos los servicios del *contact center* en estado *Started*. Un servicio del *contact center* es una entidad creada en la plataforma, la cual tiene un punto de contacto (DNIS/DDI, e-mail, cuenta de Facebook, …) asociado para que los clientes entren en contacto con el *contact center* y tiene agentes asignados que reciben las interacciones provenientes de ese punto de contacto. Un servicio puede encontrarse en 4 estados diferentes (*Started*, *Suspending*, *Suspended* y *Stoped*). El hecho de únicamente seleccionar los servicios que están en estado *Started*, es porque únicamente en este estado, los agentes asociados al servicio, van a recibir interacciones provenientes del servicio para poder tratarlas.
    - StatisticalDataProvider: Service;
    - RowsIDsDefinition: StartedServices.
- **GetStartedInboundServices**: método similar al anterior, pero que únicamente nos retorna información para los servicios de tipo Inbound.
    - StatisticalDataProvider: ServiceInbound;
    - RowsIDsDefinition: StartedServices.

- **GetStartedOutboundServices**: método similar al anterior, pero que únicamente nos retorna información para los servicios de tipo Outbound.
    - StatisticalDataProvider: ServiceOutbound;
    - RowsIDsDefinition: StartedServices.
- **GetStartedOutboundPredictiveServices**: método similar al anterior, pero que únicamente nos retorna información para los servicios de tipo Outbound.
    - StatisticalDataProvider: ServicePredictive;
    - RowsIDsDefinition: StartedServices.
- **GetAgentsSignedIn**: método para obtener información de todos los agentes que están en estado *SignedIn*, o sea, que se encuentran con su sesión de aplicación de agente abierta.
    - StatisticalDataProvider: Agent;
    - RowsIDsDefinition: AgentsSignedIn.
- **GetQueueInteractions**: método para obtener el número de interacciones que están en la lista de espera universal de la instancia que sea indicada como parámetro de entrada. La lista de espera es considera universal ya que en ella se encuentran interacciones de todos los canales/media atendidos por el *contact center*.
    - StatisticalDataProvider: QueueSessions;
    - RowsIDsDefinition: AllSessions.
- **GetServiceAgent**: este método proporciona información de los agentes que están en estado *SignedIn* y que están asignados a servicios cuyo estado es *Started.*
    - StatisticalDataProvider: ServiceAgent;
    - RowsIDsDefinition: StartedServices;
    - RowsIDsDefinitionSecondEntity: AgentsSignedIn.
- ***GetRunningInstances***: este método tiene una particularidad en relación a los anteriores, ya que es el único al que no debemos darle contexto de la instancia del contact center que deseamos obtener información, siendo su propósito la obtención de información de todas las instancias que se encuentran en estado *Running*. Una instancia puede encontrarse en 4 estados diferentes (*Running*, *Supending*, *Suspended*, *Idle*), siendo únicamente interesante para análisis las instancias en estado *Running.*
    - StatisticalDataProvider: InstanceOverview;
    - RowsIDsDefinition: RunningInstances.

La información retornada por estos pedidos hace referencia a la instancia que sea indicada como parámetro de entrada. Este retorno consiste es una tabla de datos, la cual puede ser accedida a través del nombre de la columna.

3.3.2. OCKPI: Representación de KPIs del Contact Center

La clase abstracta OCKPI define los métodos necesarios para recolección y tratamiento de los KPIs del *contact center*. El campo principal de esta clase, denominado cache, es una lista de OCSample, que es una clase auxiliar la cual define el tipo de variable donde son almacenadas cada una de las tomas de valores realizadas al KPI. Tanto la clase *OCKPI*, como la clase OCSample son genéricas, ya que no tienen restricción en relación al tipo asociado a los valores pertenecientes al KPI.

Esta clase está diseñada para sus métodos poder ser invocados en simultaneo por varios hilos de ejecución, ya que como será explicado en el apartado de la Aplicación de Consola, existen varios hilos interactuando con los objetos de esta clase al mismo tiempo. Para favorecer este comportamiento, la clase *OCKPI* tiene un atributo del tipo Mutex que controla el accedo al campo *cache*.

A partir de esta clase heredan varias subclases adaptadas para la recolección y manejo de datos para KPIs concretos del *contact center*. Las clases que heredan de *OCKPI* ya indican cual es el tipo de los valores asociados a ese KPI.

En relación a los métodos que se han diseñado dentro del ámbito de esta clase, se encuentran:

- **AddToCache**: sirve para añadir entradas dentro de la lista de objetos OCSample denominada *cache*;
- **GetValuesBetweenDates**: permite la obtención de la lista de objetos OCSample dentro del atributo *cache* que corresponde a tomas de valores del KPI entre dos fechas introducidas como parámetro de este método;
- **AverageBetweenDates**: retorna la media de los valores tomados en relación al KPI entre dos fechas introducidas como parámetro de este método;
- **RemoveBetweenDates**: limpia de la lista cache las entradas entre dos fechas introducidas como parámetro de este método.

En el siguiente diagrama de clases, puede ser consultada la relación entre *OCKPI* y OCSample y las diferentes clases que heredan de *OCKPI*:

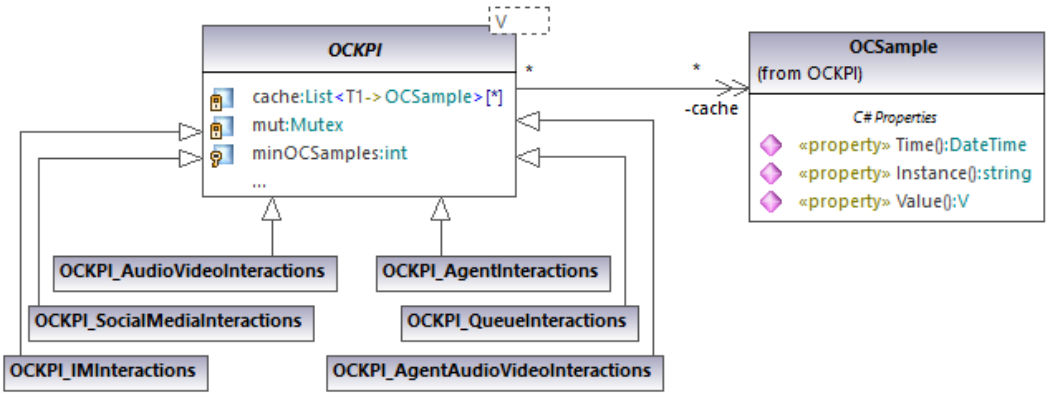

Figura 3.7: Diagrama de Clases de *OCKPI*

Los KPI utilizados en el ámbito de esta aplicación son los siguientes:

- **Interactions**: número de interacciones que están siendo atendidas por los agentes;
- **QueueInteractions**: número de interacciones en la cola de espera;
- **AgentAudioVideoInteractions**: número de interacciones de audio o vídeo siendo atendidas por agentes. Este valor ha sido también relacionado con el componente que realiza la grabación de audio, denominado OneMedia, tomando la suposición de que todas las llamadas del *contact center* son grabadas;
- **AudioVideoInteractions**: número de interacciones de audio y video que se encuentran en la cola de espera;
- **IMInteractions**: número de interacciones de tipo *instant message* que se encuentran en la plataforma de *contact center*;
- **SocialMediaInteractions**: número de interacciones de tipo *instant message* que se encuentran en la plataforma de *contact center*.

3.4. Aplicación de Consola

La aplicación de consola tiene como objetivo la recolección de datos en tiempo real, tanto de PRTG, como de la plataforma de *contact center*, gracias a los conectores implementados, para realizar el cruzamiento de esta información y obtener el consumo de recursos sobre cada componente de la plataforma de *contact center* que realiza cada instancia.

Esta aplicación puede ser ejecutada como aplicación de consola, aunque sería preferible la creación de un servicio Windows para una ejecución en segundo plano. En el caso de ser ejecutada como aplicación de consola no necesita ningún argumento de entrada ya que todos los parámetros de configuración son obtenidos del fichero app.config. También destacar que cuando la aplicación es ejecutada en la consola no da ningún *feedback* en la propia consola, siendo toda la información escrita en los ficheros de *logs*. La única interacción con la aplicación que está contemplada es pulsando CTRL+C, cuyo evento es capturado y debidamente tratado para realizar una salida limpia de la ejecución.

Para gestionar de manera unificada la información proveniente de ambas plataformas, PRTG e *contact center*, fue diseñada una clase llamada OCSys, la cual incluye una lista de los dispositivos donde están instalados los componentes del *contact center* y todos los KPIs utilizados para realizar los análisis de consumo de recursos en los componentes del *contact center*.

Esta aplicación, al iniciar crea un hilo por cada instancia de la plataforma de *contact center*, quedando el hilo del proceso principal responsable por obtener los datos de PRTG y realizar el mapeo de los datos.

El hilo principal, después del paso nombrado en el parágrafo anterior, obtiene la lista de dispositivos que forman parte de la plataforma de *contact center* gracias a un grupo previamente configurado en PRTG. En este proceso, si detecta algún sensor nuevo que no haga parte de la lista de sensores que están registrados en la aplicación, realiza ese registro para que el nuevo sensor haga parte del universo de análisis.

Cada cierto tiempo, parametrizado a través del fichero de configuración de la aplicación, el hilo principal consulta a PRTG si tiene nuevos valores para los sensores, en caso afirmativo actualiza los valores de todos los sensores y realiza los cálculos de consumo de recursos por componente que realiza cada instancia, para registrar esa información en la base de datos de la aplicación.

Para realizar los cálculos de consumo de recursos por instancia es aplicada la siguiente formula:

$$Component\ Use\ per\ Instance = \frac{Component\ Use * KPI\ Average\ per\ Instance}{KPI\ Average}$$

Donde:

- ***Component Use***: es el valor del sensor PRTG asociado al servicio Windows del componente de la plataforma de contact center;
- ***KPI Average per Instance***: es la media del valor del KPIs del *contact center*, en relación a una cierta instancia, durante el periodo de tiempo asociado a la última muestra de valores obtenida de PRTG;
- ***KPI Average***: es la media del valor del KPIs del *contact center* durante el periodo de tiempo asociado a la última muestra de valores obtenida de PRTG.

Para saber que KPI debe ser relacionado con cada componente del *contact center*, es necesario entender bien cuál es la función exacta del componente, a partir de ese conocimiento es posible entender que factores pueden influenciar un aumento o reducción de recursos por parte de ese componente. Estos factores nos van a definir el KPI más indicado para realizar el análisis. Por ejemplo, sabiendo que OneSocialConnector es el componente encargado de manejar las interacciones de redes sociales en el *contact center*, podemos directamente asociarle al KPI OCKPI_SocialMediaInteractions y con esto, poder calcular el consumo de recursos generados sobre este componente por cada instancia.

En la siguiente tabla se encuentra el mapeo entre componentes y KPIs de la plataforma de *contact center*:

Componente	KPI
OneContactBaseService	Interactions
OneContactService	Interactions
OneContactScripting	QueueInteractions
OneContactCallControl	AudioVideoInteractions
OneContactNotifier	Interactions
OneContactTimer	Interactions
OneProxy	Interactions
OnePark	AudioVideoInteractions
OneMedia	AgentAudioVideoInteractions
OneSIPConnector	Interactions
OneContactIMGateway	IMInteractions
OneSocialConnector	SocialMediaInteractions

Tabla 3.1: Mapeo entre Componentes del Contact Center y KPIs

La descripción de cada uno de los componentes de la plataforma de *contact center* puede ser consultada en el anexo "**A**
Componentes de la Plataforma de Contact Center".

Tanto los valores obtenidos de los KPIs, como los cálculos realizados en relación al uso que cada instancia hace los componentes del *contact center*, son guardados en una base de datos con la siguiente estructura:

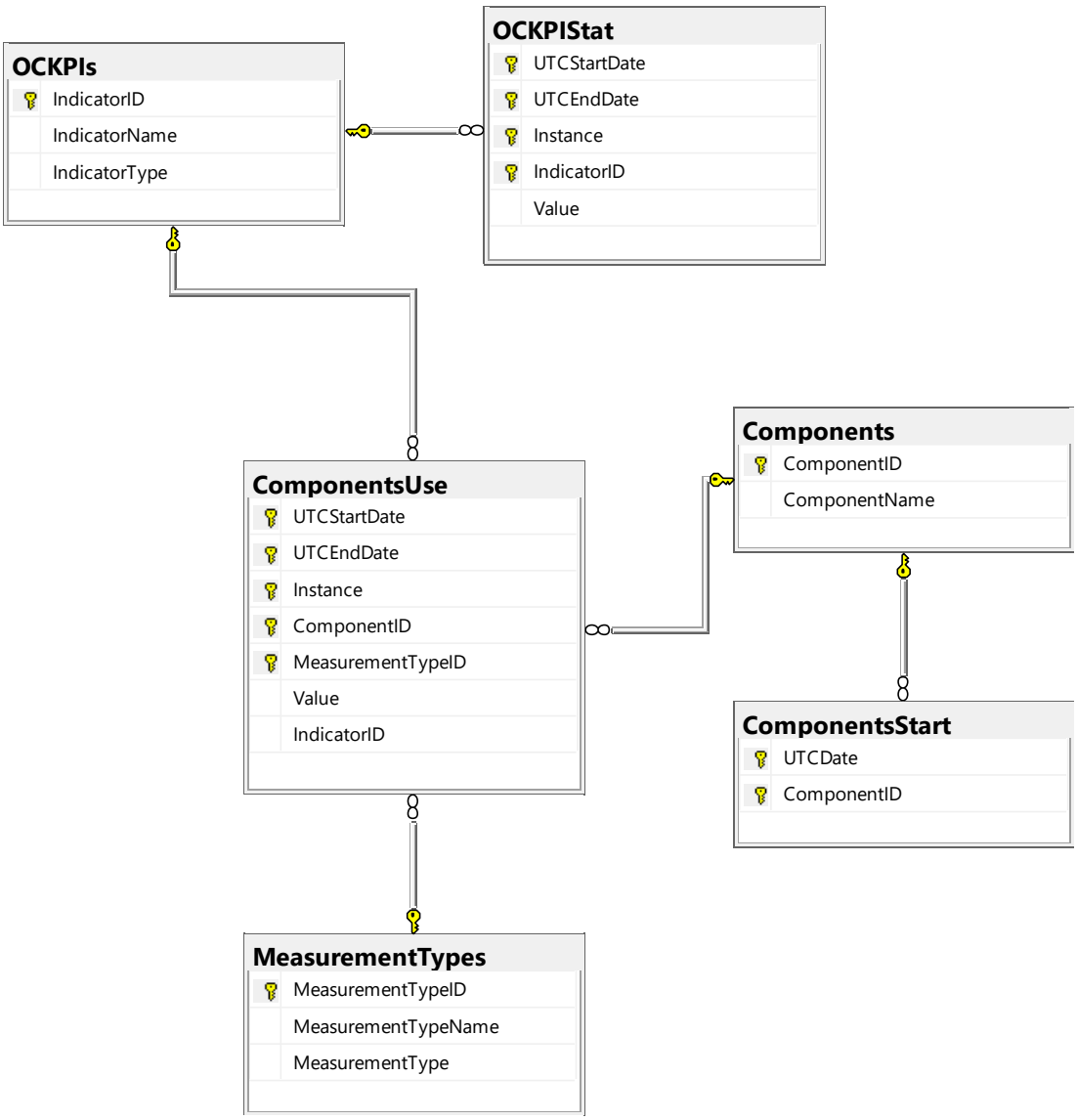

Figura 3.8: Diagrama Entidad Relación de la Base de Datos

La información contenida en esta base de datos va a ser consumida por la aplicación web.

La siguiente lista detalla el propósito de cada una de las tablas que hacen parte de la base de datos:

- **OCKPIs**: listado de los KPIs obtenidos de la plataforma de *contact center*;
- **OCKPIStat**: registro de los valores de los KPIs tomados de la plataforma de contact center;
- **Components**: listado de componentes de la plataforma de *contact center*;

- **ComponentsStart**: registro de componentes iniciados en el momento que comenzó la ejecución de la aplicación de consola;
- **MeasurementTypes**: lista de tipos asociados a los valores de los sensores de PRTG;
- **ComponentsUse**: registro del uso, por parte de cada instancia, en los componentes de la plataforma de *contact center*.

En el siguiente diagrama de clases están representadas todas las clases diseñadas en el ámbito de esta aplicación y sus relaciones, exceptuando las herencias de la clase PRTGSensorValues, las cuales ya fueron descritas anteriormente:

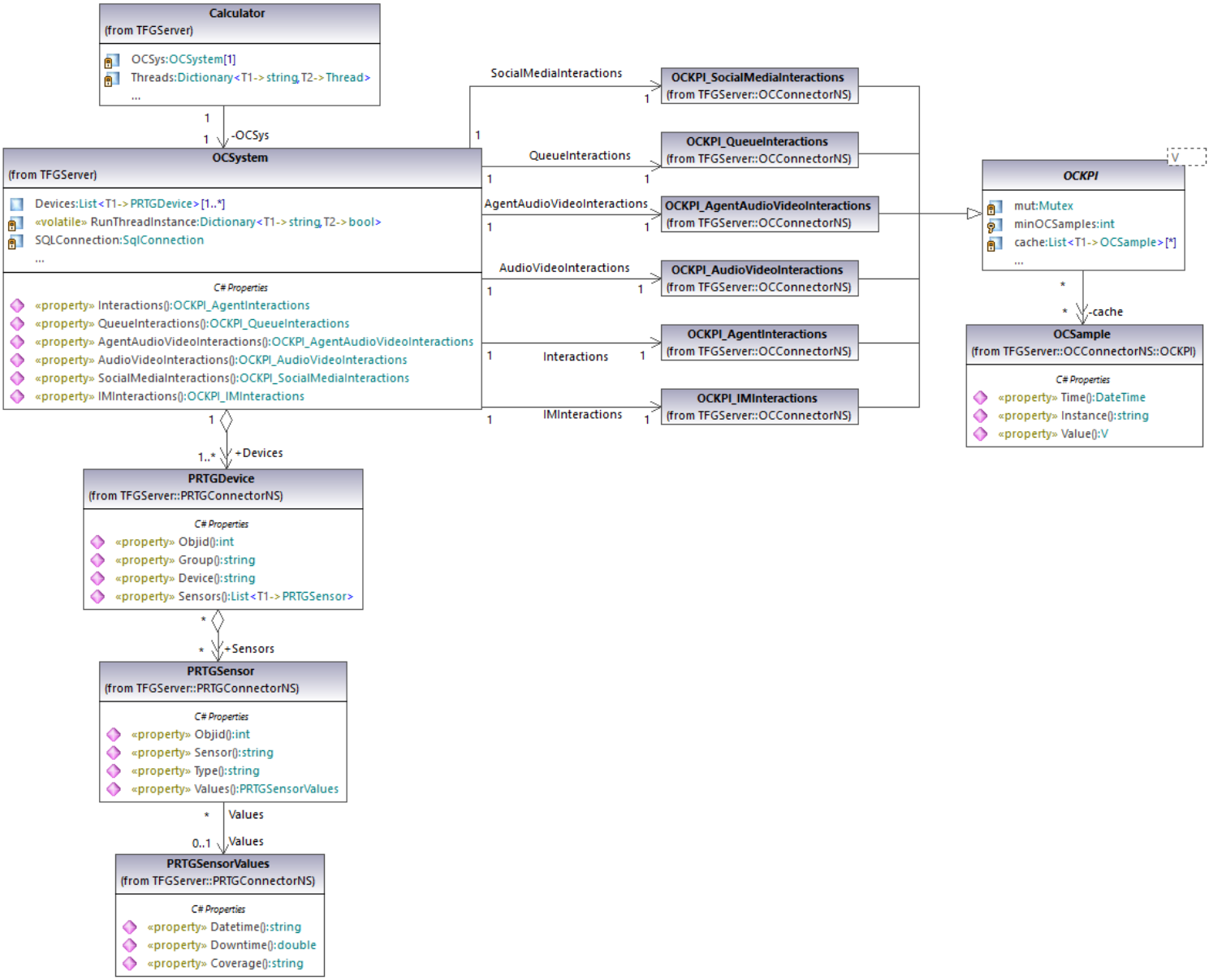

Figura 3.9: Diagrama de Clases Semi-completo (Sin Herencias de PRTGSensorValues)

3.5. Aplicación Web

La Aplicación Web diseñada tiene como objetivo principal la monitorización y la obtención de información histórica de los componentes de la plataforma de *contact center* además del uso, por parte de cada instancia del *contact center*, sobre estos componentes. Además, esta aplicación tiene un módulo que permite realizar simulaciones para saber si un nuevo cliente con su configuración específica puede ser alojado en la infraestructura actual.

Los requisitos funcionales de esta aplicación son los siguientes:

RF1: Inicio de sesión con credenciales de PRTG

Los usuarios podrán iniciar sesión en la Aplicación Web usando las credenciales de PRTG, por tanto, las tareas de gestión y creación de usuarios son realizadas directamente en PRTG.

El usuario para iniciar sesión debe indicar el nombre de usuario y la contraseña de PRTG. Caso sea perdida la conexión con el servidor, el usuario debe iniciar sesión nuevamente.

Cuando las credenciales son introducidas de manera errada, es mostrado un mensaje avisando al usuario. Pero caso las credenciales sean correctas, es mostrada la pantalla principal de la aplicación, donde es posible realizar la monitorización de los componentes del *contact center*.

RF2: Cierre de sesión

Después de que el usuario inicie sesión correctamente, su nombre aparece en la parte superior derecha de la interfaz junto al botón para cerrar sesión. Al pulsar en ese botón, el usuario es encaminado para la interfaz de login, donde deberá iniciar sesión nuevamente para acceder a la aplicación.

RF3: Monitorización de los componentes del *contact center*

Cuando el usuario inicia sesión correctamente en la aplicación, le es presentado un interfaz de monitorización de los componentes del *contact center*. En este interfaz pueden ser visualizados todos los dispositivos que forman parte de la plataforma de contact center y los componentes que están instalados en cada dispositivo.

Además, dentro de cada componente es posible visualizar los valores de los varios sensores de PRTG que están relacionados con el sensor. Estos valores van siendo actualizados de manera automática por la aplicación en intervalos de 5 minutos.

RF4: Consulta de datos históricos de consumo por instancia en los componentes del *contact center*

Dentro del menú de la aplicación existe un botón que permite el acceso a un formulario donde seleccionando los siguientes campos:

- Fecha de inicio;
- Fecha de fin;
- Componente;
- Instancia.

Es mostrada una tabla con los consumos realizados por dicha instancia en el componente del *contact center* seleccionado, durante el intervalo de tiempo escogido.

RF5: Simulación de dimensionamiento

También en el menú de la aplicación existe un botón que permite el acceso a un interfaz cuya finalidad es simular si el despliegue de un nuevo cliente sobre la infraestructura actual es viable o no, en base a las estimativas de utilización de la plataforma por parte de ese cliente.

Para obtener esta información existe un formulario en el cual, debe ser indicado el valor estimado de agentes, número estimado de interacciones tratadas de cada tipo en simultaneo por los agentes, número de interacciones en la cola de espera y si es deseada grabar las interacciones de los agentes. Con estos datos, el formulario retorna si es posible el despliegue o no.

4

Desarrollo

4.1. Sumario del Desarrollo

Tanto la Aplicación de Consola, como la Aplicación Web han sido desarrolladas utilizando el Framework .NET de Microsoft, más concretamente el lenguaje C# soportando por este Framework. Utilizando como IDE, la aplicación Microsoft Visual Studio.

En relación a las interacciones realizadas con la base de datos propia de la aplicación, todas las operaciones SQL han sido desarrolladas con utilizando *Storage Procedures*. Utilizando como motor de bases de datos Microsoft SQL Server.

Todos los módulos que hacen parte de la presente solución toman partido de la librería NLog para .NET, para un correcto registro y gestión de los *logs* generados por la aplicación. La utilización de una librería ya desarrollada para este propósito, nos proporciona la capacidad de desarrollar más rápido nuestra aplicación, pudiéndonos centrar en aspectos más importantes del proceso de desarrollo.

En este momento, los *logs* de la aplicación de consola y la aplicación web son escritos en ficheros separados, sin hacer distinción de la clase/módulo que generó ese log. Siendo actualmente registrado dentro del fichero de *logs*, la fecha y hora del log, su tipo y el mensaje asociado. Para la configuración de todo lo relacionado con NLog existe un fichero de configuración donde, entro otras cosas, es posible configurar la dirección de la carpeta donde será escrito el fichero de *logs*, el nombre de este fichero y el nivel mínimo a partir del cual queremos que sean registrados los *logs* en ese fichero [6].

Todos los parámetros de configuración utilizados por la solución, tanto en la aplicación de consola, como en la aplicación web, son obtenidos de un fichero XML de configuración, tomando partido de las funcionalidades ofrecidas por la clase ConfigurationManager del *framework* .NET [7]. A excepción del nombre de usuario y la contraseña de autenticación en PRTG, que, en el caso de la aplicación web, son introducidos a través de la propia interfaz web.

4.2. Desarrollo de la Aplicación Web con Arquitectura MVC

Para el desarrollo de la Aplicación Web descrita en el capítulo de diseño, ha sido utilizado el patrón de arquitectura Modelo-Vista-Controlador (MVC), con el propósito de separar los datos y la lógica de negocio de la aplicación, de su interfaz de usuario [8]. Este desarrollo ha sido realizado gracias al Framework ASP.NET MVC.

Aprovechando este modelo de desarrollo, la gran parte de las operaciones realizadas por la aplicación, son ejecutadas del lado del servidor, haciendo con que en el lado del cliente exista una baja carga computacional.

La estructura de la Aplicación Web desarrollada es la siguiente:

- **Modelo**:
  - AccountViewModel: define el modelo de la entidad usuario para soportar los inicios de sesión a partir de la vista Account->Login;
  - HistoricalViewModel: define el modelo para el formulario mostrado en la vista Home->Historical;
  - SimulatorViewModel: define el modelo para el formulario mostrado en la vista Home->Simulator.
- **Vista**:
  - Account (carpeta):
    - Login: formulario de inicio de sesión;
  - Home (carpeta):
    - Index: interfaz de monitorización en tiempo real de los dispositivos donde está instalada la plataforma de *contact center* y sus componentes;
    - Historical: permite la consulta de información histórica de los consumos realizados, en cada componente de la plataforma de *contact center*, por instancia;
    - Simulator: a partir de los datos estimados de utilización que un nuevo cliente va a realizar sobre la plataforma de contact center, en esta interfaz, el usuario es informado de si ese cliente podría ser alojado en la infraestructura actual o no;
    - About: vista donde se puede encontrar el resumen del TFG;
    - Contact: vista que muestra la información de contacto de los responsables de este TFG.
  - Shared (carpeta):
    - _DisplayPRTGSensorHTTP: vista de los datos de un sensor de PRTG del tipo HTTP;
    - _DisplayPRTGSensorPort: vista de los datos de un sensor de PRTG del tipo Port;
    - _DisplayPRTGSensorSIPOptionsPing: vista de los datos de un sensor de PRTG del tipo SIPOptionsPing;
    - _DisplayPRTGSensorWindowsProcess: vista de los datos de un sensor de PRTG del tipo WindowsProcess;
    - _Layout: vista que define el layout principal de la aplicación;

        - _LoginPartial: muestra en la parte superior izquierda de la interfaz, los datos de inicio de sesión y el botón de cierre de sesión;
        - Error: página de error.
- **Controlador**:
    - AccountControler: actúa de controlador para las vistas de la carpeta Account e interactúa con el conector de PRTG para permitir los inicios de sesión.
    - HomeControler: actúa de controlador para las vistas de la carpeta Home e interactúa con el conector de PRTG y realiza consultas a la base de datos alimentada por la Aplicación de Consola.
- **Librerías Externas**:
    - Conector para PRTG descrito en el capítulo "Integración con PRTG";
    - Conector para OneContact descrito en el capítulo "Integración con la Plataforma de Contact Center".

A continuación, son mostrados un par de impresiones de pantalla sobre la Aplicación Web desarrollada, la cual utiliza como base una plantilla de ASP.NET, para ayudar al lector a interpretar de manera visual las descripciones realizadas hasta el momento:

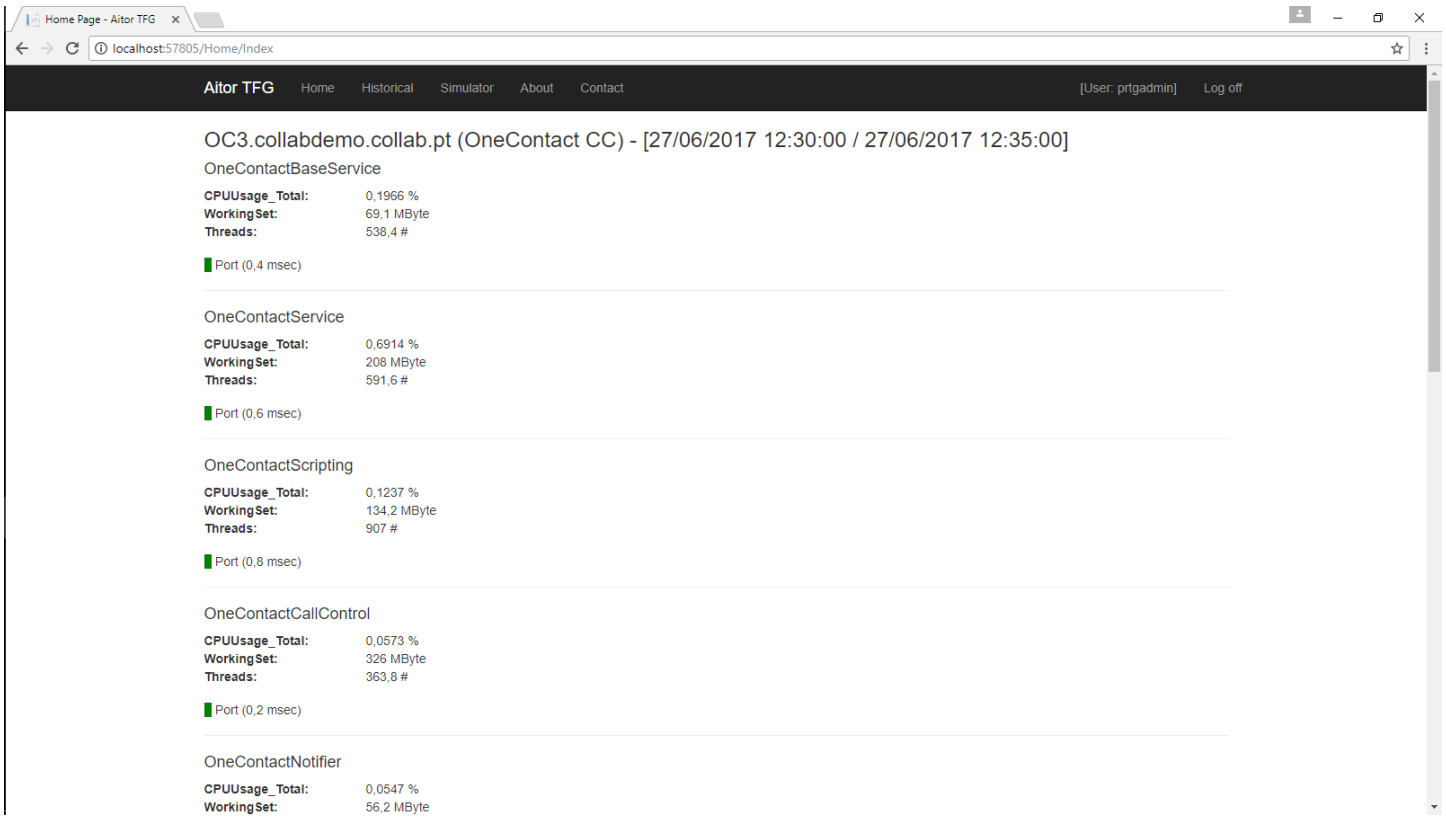

Figura 4.1: Interfaz de Monitorización de la Aplicación Web Desarrollada

Como se puede apreciar en la interfaz anterior, que además de actuar como página Index, cumple con la funcionalidad de mostrar información de monitorización en tiempo real, existe un menú en la parte superior que permite la navegación por la aplicación. En esta figura también es visible la información de usuario y el botón de logout en la parte superior derecha.

Hablando del contenido en sí, mostrado en esta interfaz, lo que se puede apreciar es la información de un dispositivo donde están instalados componentes de OneContact (el resto de dispositivos son mostrados en lista debajo de este). Por cada componente, son mostrados los valores de los diferentes sensores de PRTG, asociados a ese componente.

Al lado del nombre del dispositivo aparece el intervalo de fecha y hora al que corresponden los valores mostrados. Estos valores son actualizados automáticamente por la Aplicación Web cada 5 minutos.

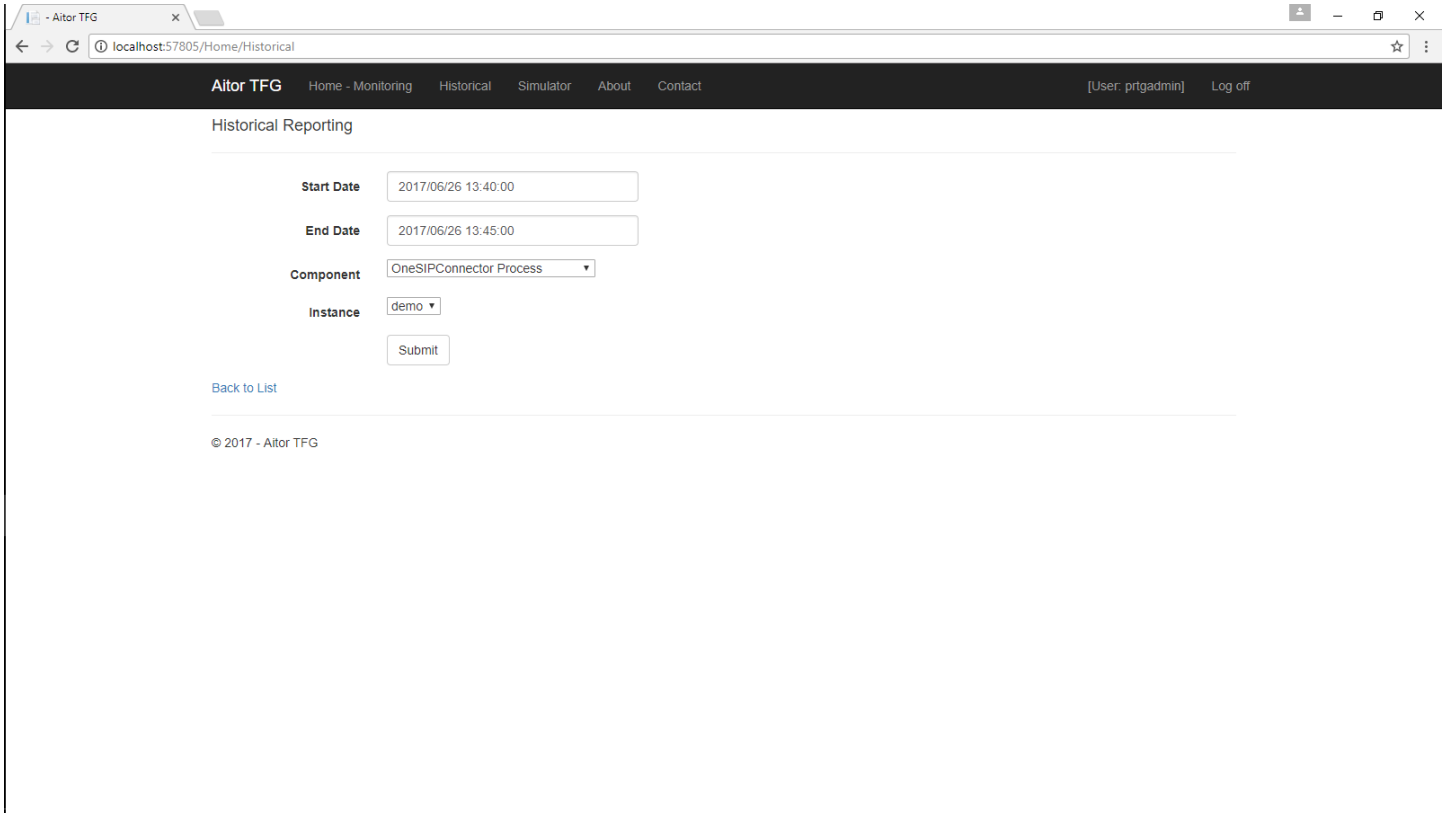

Figura 4.2: Formulario para Obtener Información Histórica de los Consumos Realizados

En la figura anterior se muestra el formulario para obtener información histórica del consumo realizado en un componente del *contact center*, por parte de la instancia seleccionada.

Al pulsar en el botón *Submit*, es presentada una tabla con los resultados referentes a la consulta realizada, los cuales son consumidos de la base de datos donde la Aplicación de Consola desarrollada en el ámbito de este TFG, registra los datos recogidos.

5

Integración, Pruebas y Resultados

5.1. Preparación del Ambiente de Pruebas

Para realizar las pruebas, tanto de la Aplicación de Consola, como de la Aplicación Web, en la fase de desarrollos y después de su finalización para la extracción de información relevante en relación a cómo realizar el dimensionamiento de una plataforma de *contact center*, ha sido necesaria la creación de un ambiente de pruebas específico.

Este ambiente de pruebas ha sido levantado sobre 4 máquinas virtuales con sistema operativo Windows Server 2012, cada una con un propósito definido:

- **VM Contact Center**: en esta máquina se encuentra alojada la plataforma de *contact center* y la plataforma PRTG que monitoriza esta misma plataforma de *contact center*;
- **VM Microsoft SQL Server**: en esta máquina ha sido alojado el motor de bases de datos que da soporte tanto a la plataforma de *contact center* como a la aplicación desarrollada en el ámbito de este TFG;
- **VM Aplicación**: en esta máquina ha sido alojada la aplicación desarrollada en el ámbito de este TFG, tanto la Aplicación de Consola (convertida a Servicio Windows), como la Aplicación Web;
- **VM Simulador**: en esta máquina es ejecutada la herramienta SIPp, cuya finalidad es la generación de trafico SIP, la cual va a simular interacciones realizadas por llamadores que entran en contacto con el *contact center* [9].

En el proceso de instalación y configuración de PRTG ha sido creado una plantilla con la configuración realizada para una posterior replicación de manera más simple y rápida. Dentro de las tareas de configuración de PRTG se encuentra la creación de cada uno de los sensores necesarios en los diferentes dispositivos que van a ser monitorizados.

Entre los sensores de PRTG requeridos en el ambiente de pruebas, se encuentran varios tipos diferentes, como, por ejemplo, sensores de procesos de Windows, sensores de puertos, sensores para aplicaciones web y una gran variedad de sensores específicos para monitorización de bases de datos. Estos sensores, como ya ha sido comentado anteriormente, se encuentran configurados dentro de la entidad dispositivo.

Cada dispositivo de PRTG, a su vez, puede estar asignado a un grupo. En el caso del ambiente de pruebas, han sido definidos dos grupos, en el primero se encuentra el dispositivo donde está instalada la plataforma de *contact center* y en el segundo se encuentra el dispositivo donde está instalado el motor de bases de datos. Esta separación viene motivada porque únicamente los dispositivos que se encuentran en el primer grupo

son analizados por la Aplicación de Consola y la definición de grupos facilita esta segmentación.

En relación a la preparación de la plataforma de *contact center* para la realización de pruebas, ha sido realizada una instalación y configuración del producto, con todos sus componentes, los cuales pueden ser consultados en el anexo "**A** Componentes de la Plataforma de Contact Center". Esta instalación ha sido realizada en una única máquina, como fue comentado en el inicio del presente capítulo.

Tanto la instalación de la plataforma de *contact center*, como la instalación de PRTG han sufrido actualizaciones durante el proceso de desarrollo del TFG. En el caso de la plataforma de *contact center*, estas actualizaciones han sido de carácter obligatorio ya que servían para corregir bug descubiertos durante el desarrollo del TFG.

La aplicación desarrollada en el ámbito de este TFG, para la realización de pruebas, ha sido instalada en una máquina aparte, como se comenta en el inicio del presente capítulo. Esta aplicación está integrada con la plataforma de *contact center* y la plataforma de PRTG nombrados anteriormente.

5.2. Pruebas y Resultados

Para la realización de las pruebas, además las máquinas comentadas en el capítulo anterior, también ha sido utilizado un PC, con Windows 10, en el cual han sido ejecutados Agent Wrappers Web que simulan la actividad realizada por los agentes del *contact center*. Los Agent Wrappers Web están compuestos por dos aplicaciones, por un lado, existe una aplicación web que implementa el Framework WebRTC y que se encarga de la componente de telefonía y, por otro lado, es ejecutada una aplicación de consola, la cual es el Agent Wrapper en sí y cuya función es manejar todos los eventos enviados por la plataforma de *contact center*.

En la plataforma de *contact center* ha sido creado un servicio de audio específico para la realización de las pruebas, cuyo objetivo es obtener valores de dimensionamiento de la plataforma de *contact center*. A este servicio, por un lado, están asignados los agentes del *contact center* que van a ser simulados por los Agent Wrappers y por otro lado vamos a ter la herramienta SIPp, comentada en el capítulo anterior, realizando llamadas para simular la actividad de las personas que entran en contacto con el *contact center*.

Para la generación de carga real en el sistema, una vez realizada la negociación SIP entre el llamador y el agente, por un lado la herramienta SIPp transmite paquetes UDP a partir de un fichero de capturas de Wireshark (.pcap), y por otro lado la aplicación web que da soporte de telefonía al Agent Wrapper Web es ejecutada en el browser Google Chrome con las *flags "--use-fake-ui-for-media-stream --use-fake-device-for-media-stream"* [10], las cuales hacen que el navegador web no pida permisiones de utilización del micrófono y la cámara, y alimentan el método WebRTC getUserMedia() con un patrón de prueba en vez de capturar la información del micrófono y de la cámara del PC.

La prueba realizada consistió en ir aumentando gradualmente el número de agentes y el número de llamadas, manteniendo siempre más o menos el mismo número de llamadas en

fila de espera (entre 5 y 10). El número pequeño de agentes al que se ha conseguido llegar con esta prueba, viene dado por las características de la máquina donde estaba alojada la plataforma de contact center, en la cual, a partir de 62 agentes tratando interacciones de voz en simultaneo, comenzaban a existir picos de consumo de CPU en la máquina que imposibilitaban la entrega de las llamadas en tiempo real a los agentes.

Para realizar una correcta interpretación de los datos que aparecen en este capítulo, se debe tener en cuenta que la máquina virtual donde está alojada la plataforma de *contact center* tiene las siguientes características:

- **Sistema Operativo**: Microsoft Windows Server 2012 Standard;
- **Tipo de Sistema**: x64-based PC;
- **Procesador**: Intel(R) Xeon(R) CPU E5-2630 0 @ 2.30GHz, 2300 Mhz, 4 Core(s), 4 *Logical Processor*(s);
- **Memoria Física Instalada (RAM)**: 8,00 GB.

A continuación, se encuentran los resultados de las pruebas de dimensionamiento de la plataforma de contacto center, cuyos valores han sido obtenidos de la Aplicación Web desarrollada en el ámbito de este TFG. Los resultados están representados en tablas, una por cada componente del *contact center*, en la cual para cada medición se indica el número de agentes que estaban atendiendo llamadas en ese momento y los 3 valores del sensor asociado al proceso Windows del componente, estos valores corresponden a la media de los valores del sensor durante el tiempo de duración de la medición. La unidad de representación de cada una de las columnas es:

- **Agents**: número de agentes tratando interacciones en simultaneo;
- **CPUUsage_Total**: porcentaje de CPU utilizado por el proceso;
- **WorkingSet**: cantidad de memoria principal ocupada por el proceso en MBytes;
- **Threads**: número de hilos que tienen como padre ese proceso.

Resultados por Componente del Contact Center

	OneContactBaseService		
Agents #	CPUUsage_Total	WorkingSet	Threads
0	0,152%	63,5	535,2
10	0,300%	65,3	539,0
20	0,595%	68,9	542,1
30	0,857%	68,1	535,6
40	0,978%	67,6	538,4
50	1,147%	67,4	539,6
60	1,415%	68,2	537,8
62	1,421%	69,6	541,2

	OneContactService		
Agents #	CPUUsage_Total	WorkingSet	Threads
0	0,128%	93,8	582,4
10	1,069%	109,2	590,6
20	1,177%	112,7	590,0
30	1,458%	114,2	588,0
40	2,122%	116,6	588,6
50	2,527%	121,7	590,4
60	2,913%	126,6	591,4
62	3,282%	131,1	589,8

Tablas 5.1 - 5.2: Resultado de las Pruebas en los Componentes BaseService y Service

	OneContactScripting		
Agents #	CPUUsage_Total	WorkingSet	Threads
0	0,040%	95,4	895,8
10	0,281%	103,8	905,4
20	0,243%	108,6	906,0
30	0,323%	112,6	906,4
40	0,483%	114,8	907,8
50	0,538%	115,8	903,6
60	0,525%	117,7	905,2
62	0,728%	119,0	905,0

	OneContactCallControl		
Agents #	CPUUsage_Total	WorkingSet	Threads
0	0,000%	305,1	361,2
10	0,546%	316,8	363,6
20	0,530%	319,2	361,7
30	0,787%	320,1	363,0
40	1,165%	320,9	361,8
50	1,268%	322,7	362,0
60	1,376%	323,0	362,6
62	1,782%	324,0	363,2

Tablas 5.3 - 5.4: Resultado de las Pruebas en los Componentes Scripting y CallControl

	OneContactNotifier		
Agents #	CPUUsage_Total	WorkingSet	Threads
0	0,003%	53,1	134,4
10	0,033%	54,4	133,4
20	0,068%	55,5	133,4
30	0,100%	56,1	133,8
40	0,119%	56,9	134,0
50	0,137%	59,9	134,0
60	0,145%	60,7	135,4
62	0,159%	60,2	135,0

	OneContactIMGateway		
Agents #	CPUUsage_Total	WorkingSet	Threads
0	0,000%	75,7	25,6
10	0,000%	74,5	25,4
20	0,000%	74,5	24,4
30	0,000%	75,2	24,4
40	0,001%	75,2	25,0
50	0,001%	75,3	25,0
60	0,003%	75,4	27,8
62	0,001%	75,4	25,6

Tablas 5.5 - 5.6: Resultado de las Pruebas en los Componentes Notifier y IMGateway

	OneSocialConnector		
Agents #	CPUUsage_Total	WorkingSet	Threads
0	0,003%	53,6	127,2
10	0,001%	54,1	125,0
20	0,003%	54,2	125,0
30	0,004%	54,3	126,0
40	0,008%	54,3	126,0
50	0,022%	54,4	126,0
60	0,020%	54,3	125,2
62	0,017%	54,3	126,0

	OneProxy		
Agents #	CPUUsage_Total	WorkingSet	Threads
0	0,000%	504,9	37,6
10	0,125%	505,2	37,2
20	0,176%	506,7	36,0
30	0,273%	507,0	36,2
40	0,339%	507,8	36,8
50	0,423%	507,6	37,4
60	0,404%	506,4	36,4
62	0,513%	506,5	36,2

Tablas 5.7 - 5.8: Resultado de las Pruebas en los Componentes SocialConnector y Proxy

	OnePark		
Agents #	CPUUsage_Total	WorkingSet	Threads
0	0,000%	101,5	44,4
10	0,449%	109,1	92,0
20	0,892%	110,1	100,3
30	0,950%	111,2	106,6
40	1,272%	112,2	101,0
50	1,484%	115,9	101,0
60	1,619%	113,3	103,6
62	1,772%	111,7	97,6

	OneMedia		
Agents #	CPUUsage_Total	WorkingSet	Threads
0	0,000%	168,0	48,4
10	2,197%	179,8	54,8
20	6,165%	192,2	63,0
30	13,000%	203,9	72,0
40	14,947%	208,2	78,2
50	15,643%	219,0	89,0
60	19,763%	228,9	97,6
62	19,904%	234,6	97,4

Tablas 5.9 - 5.10: Resultado de las Pruebas en los Componentes Park y Media

	OneSIPConnector		
Agents #	**CPUUsage_Total**	**WorkingSet**	**Threads**
0	0,000%	406,9	325,8
10	0,241%	412,4	329,6
20	2,546%	433,6	349,8
30	4,896%	434,7	353,4
40	12,413%	437,9	370,8
50	16,244%	447,2	390,6
60	18,845%	449,0	397,0
62	20,333%	450,5	399,6

Tabla 5.11: Resultado de las Pruebas en el Componente SIPConnector

En las tablas anteriores puede apreciarse que los valores de los sensores asociados a los componentes IMGateway y SocialConnector son próximos a 0 o invariable independientemente del número de agentes, los cual tiene sentido ya que las pruebas han sido realizadas únicamente para interacciones de audio.

Para entender mejor los resultados obtenidos, en los siguientes gráficos son representados cada uno de valores del sensor, para las diferentes cantidades de agentes.

CPUUsage_Total

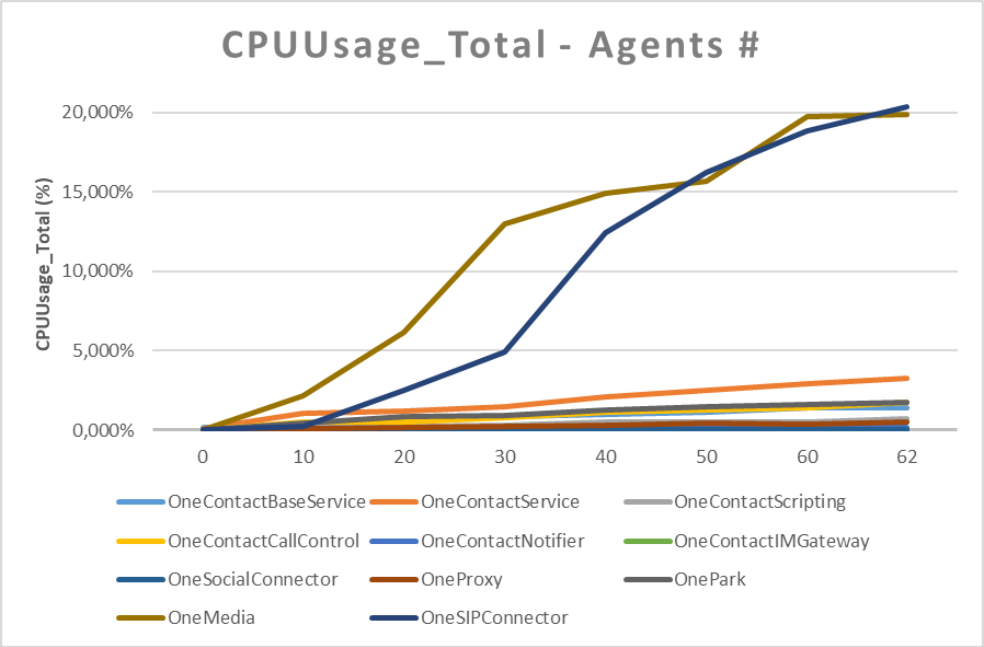

Figura 5.1: Gráfico Comparativo del Consumo de CPU de los Componentes de Contact Center [1/2]

En el gráfico anterior pude verse claramente que los dos procesos que generan una carga significativamente mayor son OneMedia, cuya función es la de grabar las llamadas, y SIPConnector, que actúa como SBC.

Para poder realizar un comparativa con mayor claridad del consumo de CPU realizado por el resto de componentes, el siguiente gráfico es igual anterior con la discriminación de los componentes OneMedia y SIPConnector.

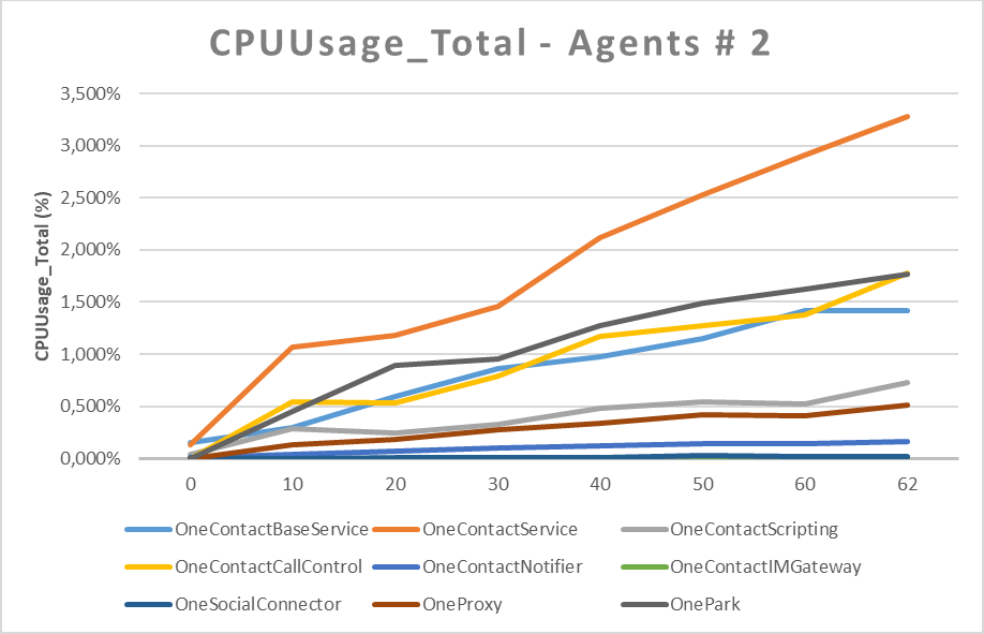

Figura 5.2: Gráfico Comparativo del Consumo de CPU de los Componentes de Contact Center [2/2]

WorkingSet

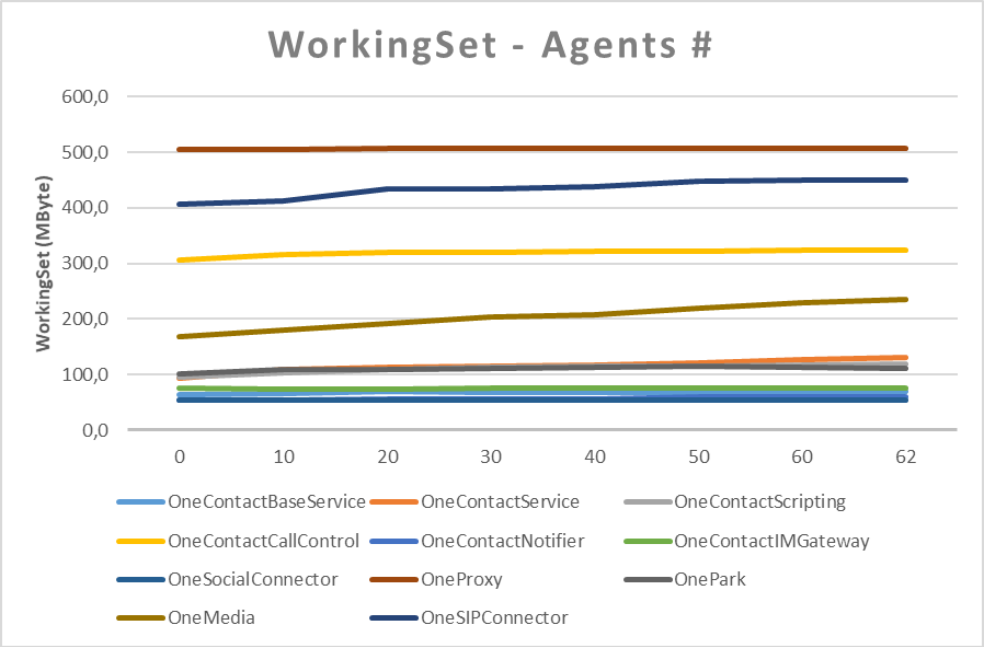

Figura 5.3: Gráfico Comparativo del Consumo de Memoria Principal de los Componentes de Contact Center [1/2]

Para la carga generada en estas pruebas, no es posible visualizar en el gráfico una variación relevante, siendo los componentes SIPConnector y OneMedia los únicos que muestran un crecimiento linear.

En el siguiente gráfico se pueden ver en más detalle la variación de consumo de memoria principal de los componentes que registran menor consumo, donde comienza a ser visible el crecimiento linear de todos los componentes, aunque en menor medida que SIPConnector y OneMedia.

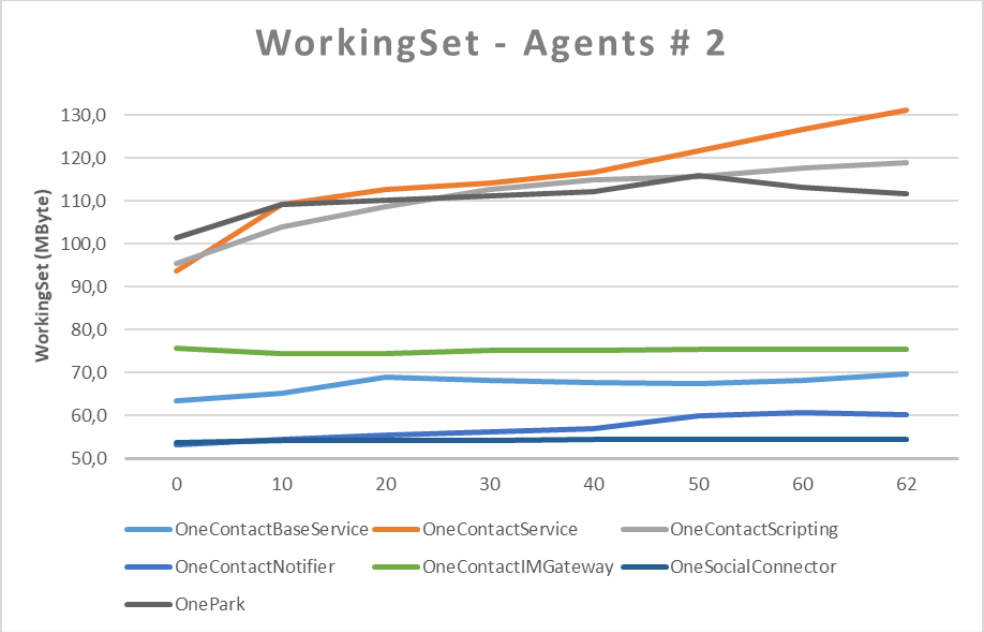

Figura 5.4: Gráfico Comparativo del Consumo de Memoria Principal de los Componentes de Contact Center [2/2]

Threads

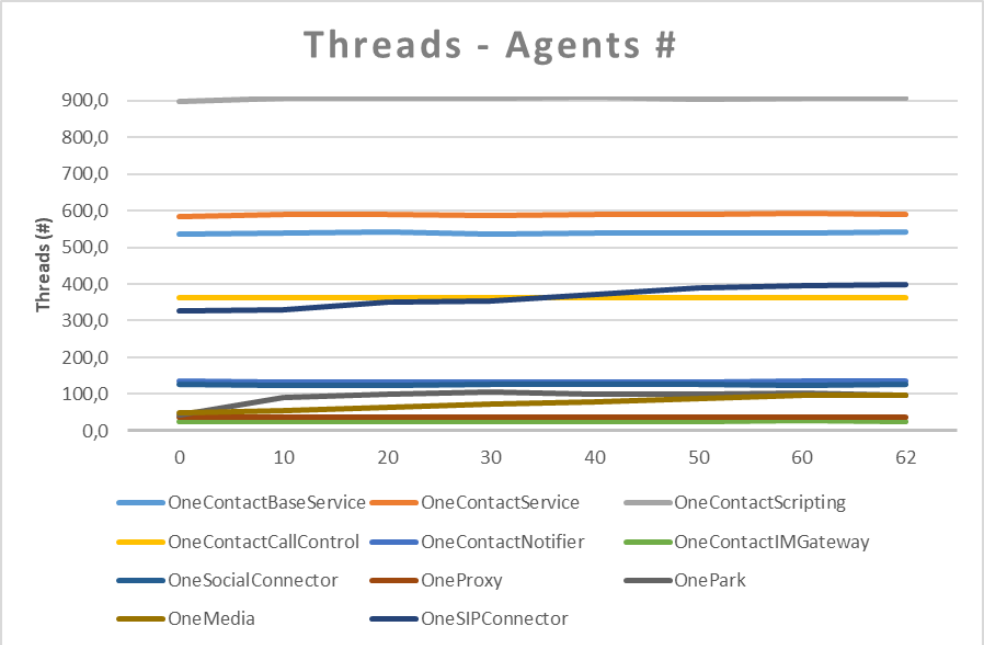

Figura 5.5: Gráfico Comparativo de Hilos en los Componentes de Contact Center [1/2]

Tanto en el gráfico anterior, donde se muestra el número de hilos creados por cada componente del *contact center* según el número de agentes que se encuentran atendiendo interacciones, como en el siguiente, donde se muestran únicamente aquellos componentes

cuyas líneas aparecen sobre puestas en el primer gráfico, es apreciable que el número de hilos se mantiene siempre constante en todos los componentes, excepto en OneMedia y SIPConnector. Únicamente en estos dos componentes del *contact center* es visible un aumento lineal del número de hilos creados por los procesos, el cual viene motivado por el aumento del número de sesiones tratadas por estos componentes.

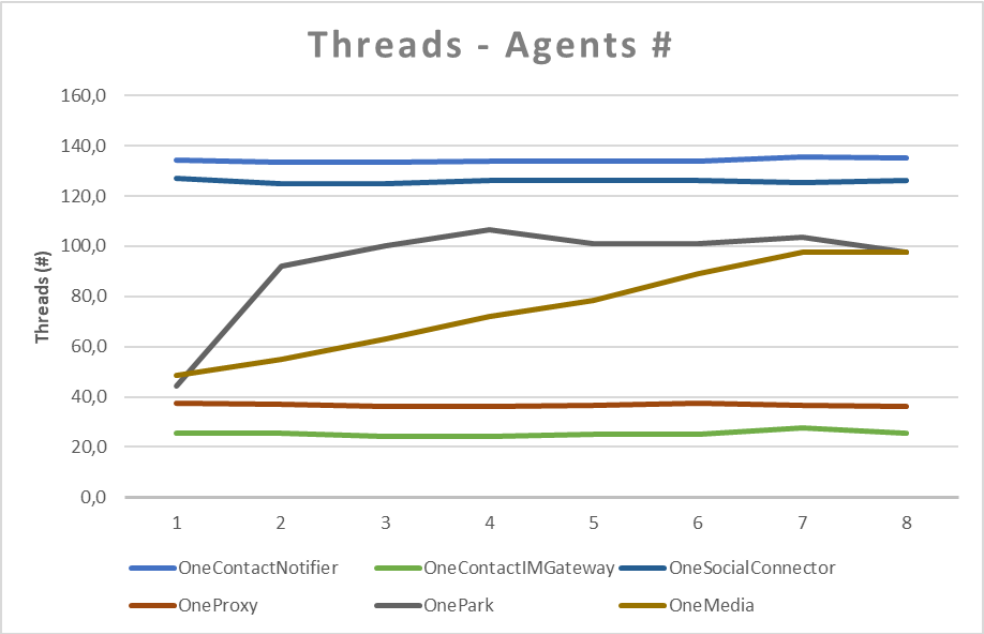

Figura 5.6: Gráfico Comparativo de Hilos en los Componentes de Contact Center [2/2]

Como nota final, transversal a los 3 indicadores del sensor, aunque no ha formado parte del ámbito de esta prueba, ya que el número de interacciones en la cola de espera se ha mantenido siempre más o menos equilibrado para el componente OnePark, se estima que el comportamiento de este componente con el aumento de carga, sería muy parecido con el comportamiento del componente OneMedia.

Resultados Totales

A partir de los resultados parciales de cada componente que forma parte del *contact center*, al realizar la suma de estos valores, se obtiene el consumo total generado por la plataforma de *contact center* en la máquina donde está instalada. Estos valores pueden ser consultados en la siguiente tabla:

Agents #	CPUUsage_Total	WorkingSet	Threads
0	0,326%	1921,4	3118,0
10	5,241%	1984,5	3196,0
20	12,393%	2036,2	3231,8
30	22,648%	2057,3	3245,4
40	33,847%	2072,6	3268,4
50	39,434%	2106,9	3298,6
60	47,027%	2123,5	3320,0
62	49,911%	2136,8	3316,6

Tabla 5.12: Consumos Totales de los Componentes del Contact Center Durante las Pruebas

Los valores de la tabla anterior, en relación a los indicadores CPUUsage_Total y WorkingSet, han sido representados en el siguiente gráfico, donde es claramente visible que partiendo del casi no consumo de recursos, hemos alcanzado niveles del 50% en relación al consumo de CPU disponible y valores ligeramente superiores al 25% de ocupación de la capacidad de la memoria principal.

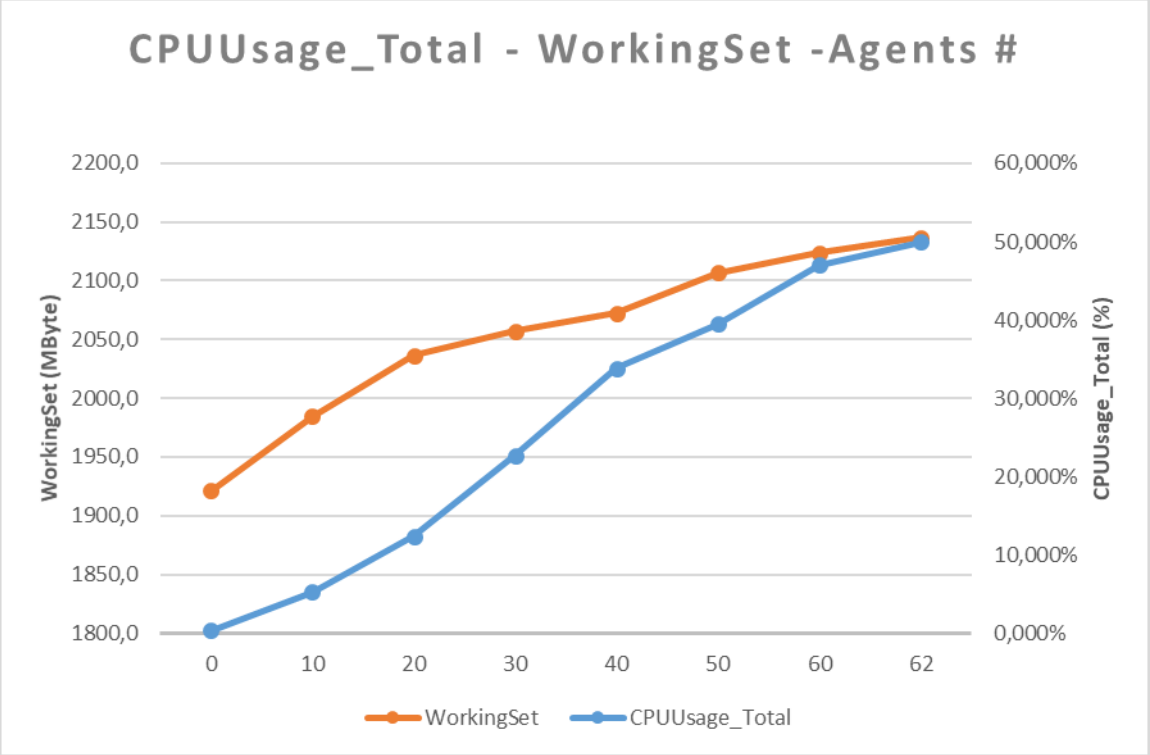

Figura 5.7: Gráfico del Consumo Total de los Componentes del Contact Center

En el gráfico anterior, es apreciable que el aumento de consumo de CPU es exponencial hasta los 40 agentes y a partir de ahí el crecimiento se vuelve más lineal. El comportamiento esperado a partir de este valor de agentes, en términos de consumo de CPU se espera que sea más o menos linear.

Como fue comentado anteriormente en términos de procesamiento de CPU no sería recomendable agregar más carga a esta máquina, ya que se corre el riesgo de que las llamadas no puedan ser entregadas a los agentes en tiempo real por falta de procesamiento.

6

Conclusiones y Trabajo Futuro

6.1. Conclusiones

La parte más importante de este trabajo para mí y de la que más conclusiones consigo obtener es la fase de pruebas, donde gracias a los desarrollos realizados en el ámbito de este TFG he podido obtener valores con los que entender mejor cual es el comportamiento real de cada uno de los componentes del *contact center*.

En relación a las conclusiones tomadas de la fase de pruebas remarcar el gran consumo de CPU realizados por los componentes OneMedia, OnePark y SIPConnector, y la gran ocupación de memoria principal por parte del SIPConnector. Estos altos niveles de consumos de recursos nos van a obligar a realizar balanceo de carga con cantidades no muy grandes de agentes, para así poder atender a esta demanda.

Por otro lado, se ha podido comprobar que para una máquina con un CPU de 4 *cores* a 2.30GHz y alrededor de 6 GB de memoria principal, en la cual estén todos los componentes del *contact center* instalados y con el motor de base de datos en otra máquina, el máximo número de agentes que podemos tener atendiendo llamadas en simultaneo es 62, con una media de 10 interacciones en la cola de espera.

Gracias a las pruebas realizadas, ha sido posible identificar patrones de consumo en los componentes, además ha sido comprobado que existen componentes que escalan de manera lineal, en cuanto otros reservan recurso en el momento del arranque y mantienen siempre el mismo consumo.

A través de los patrones de consumo es posible agrupar componentes para optimizar los servidores. Esta conclusión es particularmente importante ya que el mercado de los *contact center* se encuentra en un proceso de migración para la nube, donde la oferta está basada en *bundles*. Optimizar el alojamiento de los componentes por dispositivo va a bajar mucho el coste por hora de la plataforma.

A pesar de no haber realizado pruebas en laboratorio, tuve la oportunidad de ver informes de consumo de 24 horas donde *contact centers* de diferentes geografías abrían y cerraban su servicio en diferentes franjas horarias. Ya que gran parte de la operación se encuentra en Europa y Brasil, en la franja horaria de las 02:00 a las 07:00 (hora española) hay una significativa reducción de carga. Usando la aplicación que desarrollé, al analizar las necesidades en tiempo real, es fácil colocar un *trigger* para conectar y desconectar dispositivos y ahorrar recursos, ya que en la nube se paga por hora.

Los conocimientos adquiridos en la realización de este TFG, tanto a nivel de la plataforma de *contact center*, como en relación a PRTG son de gran ayuda en el

desempeño de mis labores diarias en Collab. Gracias a este TFG he adquiridos conocimientos al por menor de cada uno de los componentes del *contact center*.

6.2. Trabajo Futuro

En relación a los trabajos futuros que podrían ser realizados a partir de los desarrollos realizados en este TFG me gustaría de remarcar las siguientes vertientes:

- **Integración de Varias Plataformas de Contact Center:**
  El diseño realizado en este TFG únicamente permite la interacción con una plataforma de *contact center*. Teniendo en cuenta la estrategia de Collab de crecimiento en la oferta nube, la cual incluye varias plataformas de *contact center* dispersas por el mundo, sería una evolución lógica permitir que en vez de integrar y trabajar con una única plataforma de *contact center*, fuese posible integrar con varias y trabajar con la información obtenida de todas ellas de manera consolidada.
  Esto permitiría a los administradores de la infraestructura en la nube tener siempre noción de los consumos que están siendo registrados en cada plataforma de *contact center* para poder gestionar mejor los crecimientos y seleccionar mejor en que plataforma de *contact center* instalada pueden ser alojadas nuevas instancias;
- **Optimización de los Recursos en la Nube:**
  Situándonos en el mismo escenario del punto anterior, en el que la plataforma está instalada en un proveedor de servicios en la nube, donde los recursos se pagan según su utilización. Al tener relacionados los valores del negocio con los consumos que estamos realizando en la plataforma, podría ser creada una lógica en la cual las máquinas se van conectando y desconectando a medida que va siendo necesario;
- **Perfiles de Utilización:**
  A partir de los datos generados por la Aplicación de Consola sobre los consumos realizados por cada instancia, en los diferentes componentes del *contact center*, podrían ser creados patrones de utilización, con los que poder, por un lado, clasificar las diferentes instancias y, por otro lado, prever cuáles serán las necesidades en términos de recursos del dispositivo, al incluir una nueva instancia que cumple con un perfil ya existente.

Referencias

[1] Collab, «OneContact CC Overview,» [En línea]. Available: https://support.collab.com/hc/en-us/articles/218744258. [Último acceso: 26 Junio 2017].

[2] Paessler AG, «PRTG API (Application Programming Interface),» [En línea]. Available: PRTG instalado en el ambiente de pruebas. [Último acceso: 26 Junio 2017].

[3] Newtonsoft, «Serializing and Deserializing JSON,» [En línea]. Available: http://www.newtonsoft.com/json/help/html/SerializingJSON.htm. [Último acceso: 26 Junio 2017].

[4] P. AG, «GETTING PROPERTY OR STATUS OF MULTIPLE OBJECTS,» [En línea]. Available: PRTG instalado en el ambiente de pruebas. [Último acceso: 26 Junio 2017].

[5] Collab, «Using OneContact SDK,» [En línea]. Available: https://support.collab.com/hc/en-us/articles/222212087-Code-hints-and-best-practices. [Último acceso: 26 Junio 2017].

[6] NLog , «Getting started with NLog,» 17 Marzo 2017. [En línea]. Available: https://github.com/nlog/nlog/wiki/Tutorial. [Último acceso: 26 Junio 2017].

[7] Microsoft, «ConfigurationManager Class - Remarks,» [En línea]. Available: https://msdn.microsoft.com/en-us/library/system.configuration.configurationmanager(v=vs.110).aspx#Remarks. [Último acceso: 26 Junio 2017].

[8] Microsoft, «ASP.NET MVC 5,» [En línea]. Available: https://docs.microsoft.com/en-us/aspnet/mvc/mvc5. [Último acceso: 26 Junio 2017].

[9] R. GAYRAUD, O. JACQUES, R. Day y C. P. Wright, «SIPp reference documentation,» 20 Junio 2014. [En línea]. Available: http://sipp.sourceforge.net/doc/reference.html#Foreword. [Último acceso: 26 Junio 2017].

[10] WebRTC, «Testing,» [En línea]. Available: https://webrtc.org/testing/. [Último acceso: 26 Junio 2017].

A

Componentes de la Plataforma de Contact Center

- **OneContactBaseService**: ejecuta las operaciones de arranque y parada del sistema (en ejecución, inactivo, suspenso) y apunta la localización de los componentes;
- **OneContactService**: gestiona la fila de espera y realiza la distribución de las interacciones, basándose en algoritmos de encaminamiento por competencias de los agentes, tiempo de inactividad de los agentes, tiempo de espera de la llamada, etc.
- **OneContactScripting**: lanza y controla los scripts de encaminamiento, proveyendo la lógica del script y almacenando las configuraciones;
- **OneContactCallControl**: ejecuta el control de llamadas de sistemas externos, usando el concepto SIP Back2Back User Agent;
- **OneContactNotifier**: este componente entrega las notificaciones generadas por los componentes de OneContact a los componentes a los que la notificación apunta;
- **OneContactTimer**: gestiona el cronómetro de todos los componentes estáticos de OneContact;
- **OneProxy**: incluye las funcionalidades de SIP Proxy, Registrar, Redirect and Location en una única aplicación de servidor. Es una interfaz SIP para las interacciones de entrada y salida, de todos los componentes;
- **OnePark**: permite la reproducción de ficheros multimedia de bienvenida (audio y vídeo), anuncios y música, en cuanto las llamadas están en cola de espera. Este componente es capaz de reconocer tonos DTMF, lo que permite la creación de menús de audio o vídeo para interacciones por IVR;
- **OneMedia**: asume las funcionalidades de grabación de llamadas de audio, así como las funcionalidades de conferencia y escucha directa;
- **OneSIPConnector**: ejecuta las funciones de Session Border Controller (SBC), resolviendo problemas transversales de NAT entre los puntos terminales SIP (*softphones* y servidores de media) localizados en la red del cliente y en el servidor. Sus funciones incluyen:
    - Seguridad;
    - Ataque de DoS – rechaza llamadas cuando el sistema está sobrecargado;
    - Protección contra paquetes malformados;
    - Cifrado de la señalización vía TLS;
    - Autenticación, Autorización y Accounting (AAA);
    - Interoperabilidad;

  - NAT transversal;
  - Encaminamiento de Sesión;
  - Manipulación de cabeceras SIP – copia de cabeceras;
  - Soporte a IPv4 e IPv6;
  - Transporte (TCP, UDP);
  - TCP en la interfaz externa es traducido para TCP en la interfaz interna;
  - Recibe en UDP y envía en UDP;
  - QoS;
  - Definición de bit TOS/DSCO;
  - Reserva de recursos – SIPConnector tiene un límite previamente configurado y rechaza las llamadas si el número de flujos RTP sobrepasa ese límite;
  - Control del límite de *Rate*;
  - Media Release (Anti-tromboning);
  - Servicios de Media;
  - Soporte para llamadas de voz, vídeo e IM.
- **OneContactIMGateway**: apoyo IM para interacciones de *instant messaging*, usando el protocolo SIP SIMPLE;
- **OneSocialConnector**: se encarga de realizar interfaz con las plataformas de redes sociales (Facebook y Twitter), creando nuevas interacciones en el *contact center* cuando detecta publicaciones en la cuenta de la red social configurada.

B

Lista de Sensores Monitorizados a partir de PRTG

En la siguiente lista pueden ser encontrados todos los sensores, para los que el conector para PRTG desarrollado, permite obtener datos:

- **CPULoad**: carga de CPU del dispositivo;
- **MemoryLoad**: ocupación de memoria en el dispositivo;
- **CPUProcessorQueueLength**: tamaño de la cola de procesos en el CPU;
- **Pagefile**: ocupación de memoria en la Pagefile del dispositivo;
- **DiskIO_Total**: operaciones de entrada/salida que son realizadas en el disco del dispositivo;
- Pages/sec;
- **PageReads/sec**: número de páginas leídas por segundo en el dispositivo;
- **PageWrites/sec**: número de páginas escritas por segundo en el dispositivo;
- **GlobalStorage**: ocupación de la carpeta GlobalStorage de la plataforma de *contact center* donde están alojados los scripts de encaminamiento, los ficheros de audio y vídeo para los IVRs y los *logs*.
- **PING**: ping al dispositivo;
- **CPUPrivilegedTime**;
- **CPUUserTime**;
- **MemoryPageFaults/sec**: número de fallos de página en la memoria del dispositivo por segundo;
- **FreeDiskSpace**: espacio disponible en la memoria del dispositivo;
- **IISProcess**: proceso Microsoft IIS;
- **MicrosoftHyperVNetworkAdapter**: tráfico de red en el adaptador virtual de red de la máquina virtual soportada por Hyper-V;
- **OneAdmin**: acceso a la aplicación de web de administración de la plataforma de *contact center*;
- **OneSupervisor**: aplicación de web de supervisión de la plataforma de *contact center;*
- **OneAgentWeb**: aplicación de web de agente del *contact center*;
- **OneContactBaseServiceProcess**: consumos realizados por el proceso OneContactBaseService de la plataforma de *contact center*;
- **OneContactBaseServicePort**: acceso al puerto del servicio OneContactBaseService de la plataforma de *contact center*;
- **OneContactServiceProcess**: consumos realizados por el proceso OneContactService de la plataforma de *contact center*;

- **OneContactServicePort**: acceso al puerto del servicio OneContactService de la plataforma de *contact center*;
- **OneContactScriptingProcess**: consumos realizados por el proceso OneContactScripting de la plataforma de *contact center*;
- **OneContactScriptingPort**: acceso al puerto del servicio OneContactScripting de la plataforma de *contact center*;
- **OneContactCallControlProcess**: consumos realizados por el proceso OneContactCallControl de la plataforma de *contact center*;
- **OneContactCallControlPort**: acceso al puerto del servicio OneContactCallControl de la plataforma de *contact center*;
- **OneContactNotifierProcess**: consumos realizados por el proceso OneContactNotifier de la plataforma de *contact center*;
- **OneContactNotifierPerfCounters**: contadores de performance del servicio OneContactNotifier de la plataforma de *contact center*;
- **OneContactNotifierPort**: acceso al puerto del servicio OneContactNotifier de la plataforma de *contact center*;
- **OneContactTimerProcess**: consumos realizados por el proceso OneContactTimer de la plataforma de *contact center*;
- **OneContactTimerPort**: acceso al puerto del servicio OneContactTimer de la plataforma de *contact center*;
- **OneProxyProcess**: consumos realizados por el proceso OneProxy de la plataforma de *contact center*;
- **OneProxyPerfCounters**: contadores de performance del servicio OneProxy de la plataforma de *contact center*;
- **OneParkProcess**: consumos realizados por el proceso OnePark de la plataforma de *contact center*;
- **OneParkPerfCounters**: contadores de performance del servicio OnePark de la plataforma de *contact center*;
- **OneParkSIPOptions**: acceso al puerto SIP del servicio OnePark de la plataforma de *contact center*;
- **OneMediaProcess**: consumos realizados por el proceso OneMedia de la plataforma de *contact center*;
- **OneMediaPerfCounters**; contadores de performance del servicio OneMedia de la plataforma de *contact center*;
- **OneMediaSIPOptions**: acceso al puerto SIP del servicio OneMedia de la plataforma de *contact center*;
- **OneMediaCompressorProcess**: consumos realizados por el proceso OneMediaCompressor de la plataforma de *contact center*;
- **OneContactImporterProcess**: consumos realizados por el proceso OneContactImporter de la plataforma de *contact center*;
- **OneSIPConnectorProcess**: consumos realizados por el proceso OneSIPConnector de la plataforma de *contact center*;
- **OneContactIMGatewayProcess**: consumos realizados por el proceso OneContactIMGateway de la plataforma de *contact center*;

- **OneContactIMGatewayPort**: acceso al puerto del servicio OneContactIMGateway de la plataforma de *contact center*;
- **OneContactIMGatewaySIPOptions**: acceso al puerto SIP del servicio OneContactIMGateway de la plataforma de *contact center*;
- **OneContactMonitorProcess**: consumos realizados por el proceso OneContactMonitor de la plataforma de *contact center*;
- **OneScreenGatewayProcess**: consumos realizados por el proceso OneScreenGatewayde la plataforma de *contact center*;
- **OneSocialConnectorProcess**: consumos realizados por el proceso OneSocialConnector de la plataforma de *contact center*;
- **OneSocialConnectorPort**: acceso al puerto del servicio OneSocialConnector de la plataforma de *contact center*.